Chapter 7

Representation and modeling

In the preceding chapters the focus has been on the system aspects and the organization of autonomous systems, answering issues like: “How is the problem decomposed into subproblems at different abstraction levels?” and “how do the different subsystems interact?” So, we were looking at an autonomous system in the top-down direction. Having studied that, we will look now at the design of the subsystems at a certain level. “How to obtain a required functionality, how do we create a virtual sensor, how do we obtain reactive behavior?” We will turn our view 90 degrees and see the system from its different abstraction levels.

The more advanced, higher level behaviors of intelligent autonomous systems are those that do not act directly on the raw sensory data, but on an interpretation of those data. These behaviors may still be reactive, but then on the output of some virtual sensor (for instance, providing the distance to a wall); or they may be non-reactive, requiring an internal consideration of possible alternatives. The interpretation of sensory data requires its storage in an internal representation, and its processing using an internal model. This chapter treats specific examples and common threads in representations and models; we will extend some of those in subsequent chapters on sensing and behavior.

In literature, there is not always full agreement on how to differentiate between the terms ‘representation’ and ‘modeling’. We will discriminate them as follows, by analogy with prose: the model is the story of what might happen in some selected reality; the representation is the language in which it is told. The same representation may support many models (you can tell quite a number of stories in Dutch); and the same model may be implemented in many representations (Snow White, Sneeewitje, Blanc-Neige, Schneewittchen, etc.). No term includes the other; they are different aspects of the internalization of the world. The distinction does not coincide with that between syntax and semantics; syntax contains both representation and model, in that the grammar of a representation and its
conversions must be linked to a model; however, using model in the slightly more general meaning of: ‘sensible representation of the external real world’ gives a semantics to the representation of the model.

7.1 Representing space and spatial relationships

Some concepts from the level of representation have been proved so useful to so many different models that they have been developed in detail, to internally consistent frameworks. In the case of spatial models, the representation using linear vector spaces has shaped our thinking about the external world to an extent that the representation has almost become the model, and alternative representations are rarely considered. (Which is a pity; recently a powerful new description has been discovered called geometric algebra. We’re preparing a course on this, it is very relevant to computer science.) Space is like a story that is told in only one language, that of linear vector spaces.\footnote{Perhaps surprisingly when you first learned linear algebra, the converse is not true: linear vector spaces are extremely successful representations of a lot of models, many of which you should have encountered in your courses. But the explanation of linear algebra using spatial models is still the most intuitive to most people.}

Having said that, it is surprising how many times we benefit from imagining certain data to ‘live in a space’. In this chapter we will encounter task spaces, feature spaces, configuration spaces, and all have a role in autonomous systems as sensible representations. Often, transformation of the data spaces transforms a strange problem (such as path planning) into a familiar one (search) with a known algorithm ($A^*$ or dynamic programming). This then naturally raises the question of how to transform one spatial representation into the next, and we will deal with that.

In this chapter, we revisit representations of space, and highlight those elements from the mathematical theory that are applied most in autonomous systems. We would like to emphasize that many of the spaces that you will encounter in the context of autonomous systems are continuous (‘smooth manifolds’), or the discrete representation of spaces that are in fact continuous. Therefore the study of continuous spaces by mathematicians and mathematical physicists is relevant to computer scientists in autonomous systems. Below we briefly provide a chart of basic concepts and the mathematics they tie in to.

7.2 Examples of Spatial Representations

A space is a collection of locations, also called points, with a certain connectivity. This connectivity is a specification of which locations are considered close to each other. In the spaces we will encounter, there is often also a
specification of the distance between points along their connections. Let us
give some examples of spaces occurring in autonomous systems.

7.2.1 Translations: Euclidean space

The most intuitive example of a space is the space of positions of a point-like
object, constrained to move in a plane, such as the point of a pencil on a piece
of paper. It is quite natural to permit that point to move in any direction,
so to consider all neighboring points, in any direction, to be connected to
your present location. And it is also natural to take the distances ‘the same
in all directions’. We call this a Euclidean metric. Mathematically, we can
describe it by assigning a Cartesian coordinate system to the paper (right-
angled, and positively oriented: y-axis counterclockwise by π/2 radians from
the x-axis). The coordinate system is denoted by a basis \((e_1, e_2)\) of two unit
vectors. A vector \(p\) is associated to every point \(P\) – you can view it as the
translation required to bring a point to \(P\) from the origin (from now on we
will identify points and vectors, for convenience, although this is not totally
correct.) This vector \(p\) can then be characterized using linear additive laws
of multiples of these unit vectors as:

\[
p = \sum_{i=1}^{2} (p \cdot e_i) e_i
\]

(7.1)

The numbers \(p_1 = p \cdot e_1\) and \(p_2 = p \cdot e_2\) are the coordinates of \(p\). See
figure 7.1.

![Figure 7.1: Coordinate system with basis \((e_1, e_2)\)](image)

A metric is a function assigning a real number \(d(x, y)\) to every pair of
vectors \( \mathbf{x}, \mathbf{y} \) with the following properties:

\[
\begin{align*}
\text{positiveness} : \quad & d(\mathbf{x}, \mathbf{y}) \geq 0, \quad d(\mathbf{x}, \mathbf{y}) = 0 \iff \mathbf{x} = \mathbf{y} \\
\text{symmetry} : \quad & d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x}) \\
\text{triangular inequality} : \quad & d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})
\end{align*}
\] (7.2)

The natural metric in this case is the \textit{Euclidean metric}, in which the distance between two points \( P \) and \( Q \) (represented by their vectors \( \mathbf{p} \) and \( \mathbf{q} \)) is:

\[
d(\mathbf{p}, \mathbf{q}) = \sqrt{(\mathbf{p} - \mathbf{q}) \cdot (\mathbf{p} - \mathbf{q})} = \sqrt{\sum_{i=1}^{2} (p_i - q_i)^2}.
\] (7.3)

There is a unique connection between two points whose length is the distance between the points, and that is the straight line between the points. (Q: Can you prove this, or at least make it plausible?).

The Euclidean space is special in that the distance between two points \( P \) and \( Q \) only depends on their \textit{relative} position, not on their \textit{absolute} position. For other spaces, this may not be true, and distance formulas may change from position to position. Therefore in those spaces one treats distances only between infinitesimally close points, and obtains macroscopic distances by integration. The Euclidean space, in that view, has infinitesimal distance measure:

\[
ds = \sqrt{dx^2 + dy^2}
\] (7.4)

for an infinitesimal displacement \((dx, dy)\). Let us check that along a line from \( P \) to \( Q \) this indeed gives eq.(7.3). A parametrization for such a line is

\[
(q - p)t + p = (q_1 - p_1)t + p_1, \quad (q_2 - p_2)t + p_2 \quad \text{T},
\] (7.5)

(with \( t = 0 \) yielding \( p \), and \( t = 1 \) yielding \( q \)). It follows that

\[
dx = (q_1 - p_1)dt, \quad dy = (q_2 - p_2)dt,
\] (7.6)

and therefore the total length of the line is:

\[
\int_{t_p}^{t_q} ds = \int_{0}^{1} \sqrt{(q_1 - p_1)^2 dt^2 + (q_2 - p_2)^2 dt^2} = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2} \int_{0}^{1} dt = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2},
\] (7.7)

in agreement with eq.(7.3). Of course we have not yet proved that this is indeed the shortest distance, that would require a branch of mathematics called the calculus of variations.
7.2. EXAMPLES OF SPATIAL REPRESENTATIONS

7.2.2 Blips from radar or sonar

The mathematical representation of the same physical space can vary, and this should be understood clearly. For instance, take a radar or sonar that scans the task space we described above, and returns blips of data characterized by distance $r$ and direction $\phi$ (see figure 7.2).

![Polar coordinates](image.png)

Figure 7.2: Polar coordinates

We can translate such an $(r, \phi)$ data item into the $(x, y)$ representation we had above:

$$x = r \cos \phi, \quad y = r \sin \phi. \quad (7.8)$$

But we need not go back to that representation to compute properties of the data. For instance, if the object moves a little, we obtain slightly differing data $(r + dr, \phi + d\phi)$. Can we now compute how far the object has moved, directly from these numbers?

In order to do that, we first establish how small displacements $dr$ and $d\phi$ result in displacements $dx$ and $dy$. This is simply done by partial differentiation of eq. (7.8):

$$dx = \cos(\phi)dr - r \sin(\phi)d\phi, \quad dy = \sin(\phi)dr + r \cos(\phi)d\phi. \quad (7.9)$$

Note that whereas the mapping from $(r, \phi)$ to $(x, y)$ was non-linear, the differential mapping from $(dr, d\phi)$ to $(dx, dy)$ is linear (for small — in fact infinitesimal— displacements). Therefore it can be written using a matrix:

$$
\begin{pmatrix}
  dx \\
  dy \\
\end{pmatrix} =
\begin{pmatrix}
  \cos(\phi) & -r \sin(\phi) \\
  \sin(\phi) & r \cos(\phi) \\
\end{pmatrix}
\begin{pmatrix}
  dr \\
  d\phi \\
\end{pmatrix} = J
\begin{pmatrix}
  dr \\
  d\phi \\
\end{pmatrix}.
\quad (7.10)
$$

The $2 \times 2$ matrix $J$, which consists of partial derivatives of the coordinate transformation, is called the Jacobian. It is a tool to translate small displacements in one representation of the space into small changes in terms of another representation of that space, or even between spaces.

Now that we know how small displacements should be recomputed, we can compute the distance. We know that for small displacements $(dx, dy)$ the distance $ds$ is given by $ds = \sqrt{dx^2 + dy^2}$. Using matrix notation, we can
rewrite this simply (we do it for $ds^2$ so we don’t have to write the square root all the time):

$$
\begin{align*}
 ds^2 &= dx^2 + dy^2 \\
 &= \begin{pmatrix} dx \\ dy \end{pmatrix} \begin{pmatrix} dx \\ dy \end{pmatrix}^T \\
 &= (J \begin{pmatrix} dr \\ d\phi \end{pmatrix})^T (J \begin{pmatrix} dr \\ d\phi \end{pmatrix}) \\
 &= (dr \ d\phi)\begin{pmatrix}
 \cos(\phi) & \sin(\phi) \\
 -r \sin(\phi) & r \cos(\phi)
\end{pmatrix}
\begin{pmatrix}
 \cos(\phi) & -r \sin(\phi) \\
 \sin(\phi) & r \cos(\phi)
\end{pmatrix}
\begin{pmatrix} dr \\ d\phi \end{pmatrix} \\
 &= (dr \ d\phi)\begin{pmatrix} 1 & 0 \\
 0 & r^2
\end{pmatrix}
\begin{pmatrix} dr \\ d\phi \end{pmatrix} \\
 &= dr^2 + r^2 d\phi^2. \quad (7.11)
\end{align*}
$$

So the distance associated with the change in coordinate values from $(r, \phi)$ to $(r+dr, \phi+d\phi)$ is $\sqrt{dr^2 + r^2 d\phi^2}$. This formula is a lot simpler than computation through transforming both points back to $(x, y)$ and then computing the distance. But note that the formula only holds for small displacements, since for larger displacements the linearization of eq.(7.10) does not hold. (This linearization is the first term in the Taylor expansion, and for larger displacements you will need more terms to keep the error sufficiently small.) Larger distances need to be integrated from sufficiently small steps, see below in eq.(7.14). (But the same is of course true in Euclidean spaces, for instance when you want to know the circumference of a curved object.)

You see that characterization of the space and the events in it can be done in several ways. You may think of the $(x, y)$ representation as ‘natural’ — but for sensory processing of sonar data, the $(r, \phi)$ representation is just as legitimate. It is only when you need to merge data from many different sensors that you may find the need to convert all data to one common representation, and then the good old Euclidean $(x, y)$-based representation is an obvious choice. As long as you stay within the data processing of one particular sensor, you don’t have to do the conversion.

Do we call $(x, y)$-space and $(r, \phi)$ space different spaces, or different characterizations of the same space? The terminology is not quite clear here, but the consensus seems that we view it as the latter. But if we had defined the metric in $(r, \phi)$-space to be $\sqrt{dr^2 + d\phi^2}$, then physicists would have said this is a different space since the distance measures in $(x, y)$ and $(r, \phi)$ now no longer agree. Mathematicians might view the spaces with $\sqrt{dx^2 + dy^2}$ and $\sqrt{dr^2 + d\phi^2}$ as the same prototypical space (a Euclidean space), since the variables are, to them, just dummy variables. But what’s in a name?
7.2. EXAMPLES OF SPATIAL REPRESENTATIONS

Just learn the tools to convert space representations, and your programs will work, no matter what you call those spaces...

7.2.3 The configuration space of a robot arm: Riemannian space

For a robot arm, we are usually interested in the location of the object it holds, rather than the pose of the arm that holds it. The natural description of the system to solve path planning problems, however, is the configuration space of the arm. This configuration space consists of the joint positions, the so-called D-H parameters of the robot arm. Requirements like ‘minimum distance of the end effector’, which are Euclidean in the task space (which is actually the configuration space of the object) now induce a non-Euclidean metric in the configuration space of the arm. An example of this is a two-link arm with limbs of length $\ell_1$ and $\ell_2$, characterized by D-H parameters $\theta_1$ and $\theta_2$. Use figure 7.3 to derive the expression:

$$
\begin{pmatrix}
  x \\
  y
\end{pmatrix}
= 
\begin{pmatrix}
  \ell_1 \cos(\theta_1) + \ell_2 \cos(\theta_1 + \theta_2) \\
  \ell_1 \sin(\theta_1) + \ell_2 \sin(\theta_1 + \theta_2)
\end{pmatrix}

$$

(\text{where } \ell_1 \text{ and } \ell_2 \text{ are the lengths of the joints). Then we find the Jacobian by partial differentiation:}

$$
\begin{pmatrix}
  dx \\
  dy
\end{pmatrix} = 
\begin{pmatrix}
  -\ell_1 \sin(\theta_1) - \ell_2 \sin(\theta_1 + \theta_2) & -\ell_2 \sin(\theta_1 + \theta_2) \\
  \ell_1 \cos(\theta_1) + \ell_2 \cos(\theta_1 + \theta_2) & \ell_2 \cos(\theta_1 + \theta_2)
\end{pmatrix}
\begin{pmatrix}
  d\theta_1 \\
  d\theta_2
\end{pmatrix}
\quad J
\begin{pmatrix}
  d\theta_1 \\
  d\theta_2
\end{pmatrix}

$$

Just as in the example before (eq.(7.11)), the metric in the Euclidean space is now determined as the matrix $J^T J$ (sometimes called the metric tensor), which yields (check this!):

$$
ds^2 = \begin{pmatrix}
  dx \\
  dy
\end{pmatrix}
\begin{pmatrix}
  dx \\
  dy
\end{pmatrix} = 
(d\theta_1 \ d\theta_2) J^T J \begin{pmatrix}
  d\theta_1 \\
  d\theta_2
\end{pmatrix}
= \begin{pmatrix}
  \ell_1^2 + \ell_2^2 + 2 \ell_1 \ell_2 \cos \theta_2 \\
  \ell_2^2 + \ell_1 \ell_2 \cos \theta_2
\end{pmatrix}
\begin{pmatrix}
  d\theta_1 \\
  d\theta_2
\end{pmatrix}
= (\ell_1^2 + \ell_2^2 + 2 \ell_1 \ell_2 \cos \theta_2) d\theta_1^2 + 2(\ell_2^2 + \ell_1 \ell_2 \cos \theta_2) d\theta_1 d\theta_2 + \ell_2^2 d\theta_2^2
$$

(7.12)

This formula implies that a small angular motion when the robot is folded ($\theta_2 = \pi \rightarrow \cos \theta_2 = -1$) covers a lot less distance than when the robot is stretched out ($\theta_2 = 0 \rightarrow \cos \theta_2 = 1$). See figure 7.3. As a consequence, the path that realized the minimum distance between two configurations (points in configuration space) is a curve, with the angles changing in just the right way to minimize the end effector distance, i.e.: to make the end effector move along a straight line in the task space.

The actual computation of the length of a path requires the summation of a lot of infinitesimal distances along it, or (going to the continuous space
7.2.4 Metric spaces in general

We have now seen enough examples to summarize the general case of a space with a Riemannian metric, i.e. a ‘curved space’.

A point in such a space is denoted by a vector, which may be characterized by coordinates \( \mathbf{x} = (x_1 \ x_2 \ \cdots \ x_n) \). In smooth spaces a small change is denoted by the form: \( d\mathbf{x} = (dx_1 \ dx_2 \ \cdots \ dx_n) \), where there are \( n \) components in an \( n \)-dimensional space. If we then have for the length \( ds \) of a small element:

\[
ds^2 = d\mathbf{x}^T M d\mathbf{x} \quad (= \sum_{i,j} M_{ij} dx_i dx_j),
\]

we call \( M \) the metric tensor of a Riemannian metric. The expression in brackets is the calculation you have to perform in your implementation, which is just the basic matrix multiplication. (Q: what is the metric tensor for a Euclidean space?) (Q: what is \( M_{12} \) in the example of the robot arm?)

These infinitesimal distances can be integrated along a curve \( C \) connecting two points \( p \) and \( q \) (because integration is like summing infinitesimals). Just parametrize the curve by some parameter \( t \) as \( C(t) \) with \( C(t_p) = p \) and \( C(t_q) = q \). As a consequence, the \( d\mathbf{x} \) becomes \( \frac{d\mathbf{x}}{dt} dt \) and so the length \( L_{pq}(C) \) is given by:

\[
L_{pq}(C) = \int_{t_p}^{t_q} ds = \int_{t_p}^{t_q} \sqrt{\left(\frac{d\mathbf{x}}{dt}\right)^T M \left(\frac{d\mathbf{x}}{dt}\right)} dt
\]

\[
= \int_{t_p}^{t_q} \sqrt{\sum_{i,j} M_{ij} \frac{dx_i}{dt} \frac{dx_j}{dt}} dt. \quad (7.14)
\]

Verify that eq.(7.3) and its derivation provide an exact example of this general definition. What is the metric tensor in this case?
7.2. EXAMPLES OF SPATIAL REPRESENTATIONS

Figure 7.4: Minimum length paths in the configuration space of a 2-link arm, under a Euclidean metric (b) and a Riemannian metric (d); (a) and (b) show the corresponding arm motions.
The distance between $\mathbf{p}$ and $\mathbf{q}$ is the infimum (greatest lower bound) of the length $L_{pq}(C)$ over all curves connecting $\mathbf{p}$ and $\mathbf{q}$:

$$d(\mathbf{p}, \mathbf{q}) = \inf_C L_{pq}(C).$$

(7.15)

(It is often impossible to get the formula in closed form; when you implement it, use the principles of dynamic programming to do the optimization locally, so that you do not actually have to compute all the paths.) It is customary to denote a space of which the metric tensor is given as: ‘a space with a Riemannian metric’, even though actually computing the metric satisfying the metric properties equation 7.2 would still require integration and minimization. It turns out that as long as the metric tensor is a positive definite symmetric matrix, the metric properties are automatically satisfied; so that is the only thing we need to check to see whether we actually have a metric, given the metric tensor.

It is customary to denote a space of which the metric tensor is given as: ‘a space with a Riemannian metric’, even though actually computing the metric satisfying the metric properties eq.(7.2) would still require integration and minimization. It turns out that as long as the metric tensor is a positive definite symmetric matrix, the metric properties are automatically satisfied; so that is the only thing we need to check to see whether we actually have a metric, given the metric tensor.

Riemannian metrics occur everywhere. For instance, if you have to plan the fastest path for a vehicle over a terrain with mud, then the distance measure you are interested in computing is the time $t$. In the muddy parts, the velocity of the car will be lower, so the time it takes to take a small step $(dx, dy)$ will be longer. This would be described by a Riemannian metric given the time $dt$ taken as $dt^2 = (v_x dx)^2 + (v_y dy)^2$. Note that $v_x$ and $v_y$ are functions of $x$ and $y$: this is a space-variant metric.

In general metric spaces (without obstacles), there is always a path that realizes the minimum distance between two points $\mathbf{p}$ and $\mathbf{q}$, so we can still give a meaningful definition to what we called a straight line between $\mathbf{p}$ and $\mathbf{q}$. Since it is curved in the sense of non-linear in the coordinates, people prefer to call such a minimum-length path a geodesic. Figure 7.4 shows the difference in using the Euclidean metric, or the Riemannian metric, in planning a path for the 2-link arm.

### 7.2.5 Transforming spaces

As we have seen in the examples that went before, we often wish to make a transition in coordinates describing the same physical space. If we perform a change of coordinates from the coordinates $x_i$ to new coordinates $X_j$, this is done by a transformation formula. If we denote the coordinate vectors by $\mathbf{x}$ and $\mathbf{X}$ this is:

$$\text{coordinates transform as : } \mathbf{x} \rightarrow \mathbf{X} : \quad X_i = f_i(x)$$

(7.16)
small changes transform as: \[ d\mathbf{X} = Jd\mathbf{x}, \] (7.17)

where \( J \) is the Jacobian matrix, of which the \( i, j \) component is:

\[ J_{ij} = \frac{\partial f_i}{\partial x_j}. \] (7.18)

The metric tensor \( M' \) in the space characterized by \( \mathbf{X} \) is related to the metric
tensor \( M \) in the \( \mathbf{x} \)-characterization, for it is implicitly defined by:

the metric transforms as: \[ d\mathbf{X}^T M' d\mathbf{X} = d\mathbf{x}^T (J^T M' J) d\mathbf{x} \equiv d\mathbf{x}^T M d\mathbf{x}. \] (7.19)

So, for the metric tensors, we have the transformation law:

\[ M = J^T M' J, \quad M' = (J^{-1})^T M J^{-1}, \] (7.20)

(assuming that \( J \) is invertible). (Q. Prove that in a Euclidean space, a
change of coordinates to a new coordinate system that differs from the old
one only by a translation and a rotation, has no effect on the formula for
the metric. Really do this, it is a good exercise to make sure you have
understood everything!)

### 7.2.6 Feature spaces

Suppose you have to classify two classes of rectangles of which you have seen
many examples. One class appears to be more square than the other, so you
decide to characterize the rectangles by their length \( \ell \) and width \( w \). This
defines a space of rectangles, with coordinates \( (\ell, w) \). We call this a feature
space. It is not yet clear what the metric should be, but we are going to find
out.

We begin by plotting the data (see figure 7.5). It falls into two clusters,
one for class \( A \) and one for class \( B \). It is so regular that it makes
sense to view the variation as ‘noise’ around ideal representatives for those
classes. And with the data as sketched, a very sensible estimate of that ideal
representative is ‘center of gravity’ of the distribution.

Let us denote a typical point belonging to class \( A \) as \( (\ell_i, w_i)_A \), and a
typical point belonging to class \( B \) as \( (\ell_i, w_i)_B \). Then good estimators for
representatives \( \hat{A} \) and \( \hat{B} \) are the means of the measured data:

\[ \hat{A} = \frac{1}{n_A} \sum_{i=1}^{n_A} (\ell_i, w_i)_A \equiv (\bar{\ell}_A, \bar{w}_A), \] (7.21)

\[ \hat{B} = \frac{1}{n_B} \sum_{i=1}^{n_B} (\ell_i, w_i)_B \equiv (\bar{\ell}_B, \bar{w}_B). \] (7.22)
where $n_A$ and $n_B$ are the number of data points belonging to class $A$ and $B$, respectively.

If you would now get a new data item $X = (\ell, w)$, how would you decide whether to classify it as $A$ or as $B$? The sketch suggests that just simply taking the Euclidean distance in the feature space and assigning to the class that is closest is not really that good: it does not reflect the noise in the data too well. Class $A$ apparently has not a lot of noise in the direction $(1, -1)$, whereas class $B$ has a lot. So for the distance to $\hat{A}$, we should take a distance of $X$ to $\hat{A}$ in that direction as very serious, but for $\hat{B}$ it is much less so. So a good question in this classification problem is: how do you set up the distance measures so that you reflect this spread in the data?

The sensible answer is to use an estimate of the covariance matrix of the data sets, which characterizes the variance in the multi-dimensional parameter space. When the variance is large in a certain direction in that space, distances in that direction are less relevant. Therefore distance can be taken inversely proportional to the covariance:

$$M_A = \tilde{\text{cov}}_A^{-1}$$

with (as you may remember from your statistics classes):

$$\tilde{\text{cov}}_A = \frac{1}{n_A - 1} \sum_{i=1}^{n_A} (\ell_i, w_i)_A - \hat{A})(\ell_i, w_i)_A - \hat{A})^T$$

$$= \frac{1}{n_A - 1} \left( \begin{array}{cc} \sum_A (\ell_i - \bar{\ell}_A)^2 & \sum_A (\ell_i - \bar{\ell}_A)(w_i - \bar{w}_A) \\ \sum_A (\ell_i - \bar{\ell}_A)(w_i - \bar{w}_A) & \sum_A (w_i - \bar{w}_A)^2 \end{array} \right)$$

Figure 7.5: Rectangle space
and similarly for \( B \). Such a distance measure is called a Mahalanobis distance. We then classify \( X \) to \( A \) if the Mahalanobis distance to \( \hat{A} \) is smaller than that to \( \hat{B} \):

\[
(X - \hat{A})^T M_A (X - \hat{A}) < (X - \hat{B})^T M_B (X - \hat{B}),
\]

and to \( B \) otherwise. This has the desired properties of adjusting the classification certainty to the inherent spread in the data.

We just used this example here to show that feature spaces, too, are spaces with a Riemannian metric. If you would prefer to classify your rectangles by, say, the area \((\ell w)\) and the deviation from squareness \((\ell - w)\), then you now know how to transform your classification rule: compute the Jacobian of the coordinate transformation that the change of characterization induces, and use the metric transformation formula.

### 7.2.7 Manhattan: a discrete space

On the island of Manhattan, north of the Village, streets run in a grid pattern. For simplicity, let us assume that it is a regular grid pattern, with the length of a block between avenues equal to 50 meters, and between streets 100 meters. It makes sense to identify intersections by their street and avenue number. Since these change discretely, we call the space that results a discrete space. The formula for the length of a path between two intersections \( p \) and \( q \) is in this case:

\[
d(p, q) = 100|q_1 - p_1| + 50|q_2 - p_2|
\]  

(7.26)

This distance \( d(p, q) \) satisfies the metric properties of eq.(7.2). (Q: Show this!).

This distance is obviously not Euclidean. We can compose it from minimization over all paths made up of small steps between adjacent intersections:

\[
d(p, p + dp) = 100|dp_1| + 50|dp_2|.
\]  

(7.27)

Here \( dp_1 \) and \( dp_2 \) are discrete increments. You see that this way of considering distances as made up of lengths of the smallest possible steps is very similar to computing the length of a path in a Riemannian metric. There are some differences, notably in the use of \( \sqrt{\cdot} \) and \( \cdot^2 \), but the structure is the same.

Another example of a discrete space is a chessboard. If we consider how many moves it takes to move a king from square \( p \) to square \( q \) (both characterized by board coordinates), we find a distance measure:

\[
d(p, q) = \min\{|q_1 - p_1|, |q_2 - p_2|\}
\]  

(7.28)

Q: Show that this satisfies the metric properties.
7.2.8 Paths for a mobile robot

Let us consider a circular path for a mobile robot, see figure 7.6, and describe it in terms of the position \((x, y)\) it reaches relative to its initial position. Obviously natural parameters to characterize the arc are in terms of its radius \(R\) and the sector angle \(\phi\) (in radians).

![Diagram of a circular path](image)

Figure 7.6: Making a turn to a point \((x, y)\).

These need to be chosen to meet the point \((x, y)\), so we have:

\[
\begin{align*}
  x &= R \sin \phi \\
  y &= R(1 - \cos \phi).
\end{align*}
\]

\[
\begin{align*}
  R &= \frac{x^2 + y^2}{2y} \\
  \phi &= \arctan\left(\frac{2xy}{x^2 - y^2}\right)
\end{align*}
\] (7.29)

Especially near \(y = 0\) (straight ahead!), a small change in \((x, y)\) may cause a big change in the parameters \((R, \phi)\), since \(R\) changes from \(-\infty\) to \(+\infty\). You may verify this by computing the Jacobian; you should get:

\[
\begin{pmatrix}
  dR \\
  d\phi
\end{pmatrix} = \begin{pmatrix}
  \frac{x}{y} & \frac{y^2 - x^2}{2y^2} \\
  \frac{2y}{(x^2 + y^2)^2} & \frac{2x}{x^4 - y^4}
\end{pmatrix} \begin{pmatrix}
  dx \\
  dy
\end{pmatrix}
\] (7.30)

For this reason it is much better to characterize the path by the curvature \(\kappa \equiv 1/R\) of the circular arc that changes smoothly around the forward direction. (Q: Compute the Jacobian and show that it behaves much more stably.)

Now we need to follow this path using a robot.

When the robot is, kinematically, like a car, \(\kappa\) and \(\phi\) can be related to the control parameters: steering angle (of the wheels), and velocity (or distance), using the geometry of figure 7.7(a). For simplicity we assume that the two front wheels could be replaced by one front wheel in the middle with steering angle \(\theta\). Then:

\[
\frac{\ell}{R} = \tan \theta \quad \text{so} \quad \theta = \arctan(\kappa \ell),
\] (7.31)
with \( \ell \) the length of the car, between the axles. If the other control parameter is the velocity \( v \), then (under the assumption that the mobile robot drives at uniform speed) \( v \) can be computed from \( vt = \phi R \), so:

\[
v = R \frac{\phi}{t} = \frac{\phi}{\kappa t}.
\] (7.32)

Note that in these equations, \( R \) is the radius of the circular path described by the center of the rear axle. The center of the front axle describes a path with a different radius, namely \( R/\cos \theta \) (verify this!).

The path of the car thus has various levels of description, which can be converted into each other:

\[
(v, \theta) \leftrightarrow (\kappa, \phi) \leftrightarrow (x, y)
\] (7.33)

The first is very much related to the mechanism of the car, the second to the geometry of the path, and the third more to an objective representation of the environment, independent of whether it will be driven through or not.

Now suppose that the robot is one of those standard mobile robots which does both its driving and steering using two wheels (like a robotic wheel chair). Of course you can do your path planning for such robots combining ‘turning on the spot’ with ‘going straight to the next point’. However, such maneuvers would require stopping-and-going, and would not be very efficient. So you may just use a path planner which produces a path in terms of circular arcs and straight lines, and translate those to commands for this robot. Then we need to convert \((\kappa, \phi)\) combinations to the control parameters of this wheel chair robot.

This is sketched in figure 7.7(b). If \( \alpha_l \) and \( \alpha_r \) are the angular velocity (angle per unit time) of the left and right wheels, respectively, and the wheels

\[
\begin{align*}
\alpha_l &= \frac{\ell}{\kappa t}, \\
\alpha_r &= \frac{\ell}{\kappa t},
\end{align*}
\]
have radius $\rho$, then in a time interval $t$ they travel:

$$d_\ell = \rho \alpha_\ell t \quad \text{and} \quad d_r = \rho \alpha_r t \quad (7.34)$$

(so the velocity of the wheels is: $v_r = \rho \alpha_r$ and $v_\ell = \rho \alpha_\ell$). From the figure, with $w$ denoting the width of the car:

$$\phi(R - \frac{1}{2}w) = d_\ell \quad \text{and} \quad \phi(R + \frac{1}{2}w) = d_r \quad (7.35)$$

so that

$$\kappa = \frac{2}{w} \frac{d_r - d_\ell}{d_r + d_\ell} = \frac{2}{w} \frac{\alpha_r - \alpha_\ell}{\alpha_r + \alpha_\ell} \quad \text{and} \quad \phi = \frac{d_r - d_\ell}{w} = \frac{\rho t}{w} (\alpha_r - \alpha_\ell). \quad (7.36)$$

We thus have a chain of models on different levels, all related to each other by the above equations:

$$(\alpha_r, \alpha_\ell) \leftrightarrow (d_r, d_\ell) \leftrightarrow (\kappa, \phi) \leftrightarrow (x, y) \quad (7.37)$$

Each is a legitimate description of the path of the mobile robot, and the choice of representation is therefore not dictated by the problem, but by your way of solving it (e.g. the kind of path description you prefer), or by the level of detail that you need in spelling out the control actions. Never be afraid to switch to another representation, the simplification of your algorithms is often more than worth the price of the conversion!

Comparing the two descriptions, we see that at the $(\kappa, \phi)$-level of description, the two vehicles are kinematically the same. It thus makes sense to do path planning in an environment at this level, if you do not yet know what vehicle you will need to control. The translation of the planned $(\kappa, \phi)$-sequence can then be translated into the vehicle-specific parameters later on. (But take some care: you then have to take a worst case estimate of the vehicle size, so you may not get the optimal path for any specific vehicle. You exchange optimality for modularity, and this is very common trade-off!)

Q: The above model of course assumes that $w$ and $\rho$ are known. If they are not, they can be estimated through one’s actions, from the deviation of the desired result and the actual result achieved based on the computation using assumed values for $w$ and $\rho$. Work this out: how do the adjustments in $w$ and $\rho$ depend on the mismatch in $x$ and $y$?

### 7.3 Objects in space; free space; incomplete knowledge

The spaces we encounter with our autonomous systems are rarely empty: there are objects in them. Some of those are objects to be manipulated, others are to be avoided. Objects are commonly represented in one of two
7.3. OBJECTS IN SPACE; FREE SPACE; INCOMPLETE KNOWLEDGE

ways: as boundaries or as volumes. For non-invasive interactions with the objects, the boundary representation is sufficient, and you commonly find it in computer graphics since the interaction of light rays and objects is well approximated as a phenomenon involving only the boundary. Volume representations are required for interactions that take place inside the boundary, such as light scattering in clouds, or milling operations on solid objects – the latter explains why volumetric representations are common in CSG (computational solid geometry). You should choose the object representation that fits your needs.

However, an autonomous system needs to represent more than just the objects to represent the environment: it moves in the part of space not occupied by the objects, which we call free space. We need to represent this free space explicitly to specify our actions, and since we are moving around in it, we need an explicit volumetric description of free space.

There is an additional complication which is not present in other fields, and that is that an autonomous system typically does not have exact and complete information about the environment around it, and that therefore its internal representation of that environment should not assume exactness and completeness. Also, we should distinguish the two carefully:

- **Uncertain knowledge**
  The knowledge about the environment is uncertain, due to the accuracy and the interpretation of the sensors. Inaccuracy and digitization effects conspire to make a represented sensory datum compatible with a number of possible real world events. Typically, one would represent this using formalisms of probability and/or possibility. You would use probability when it is a simple matter of spread, or region of sensitivity. Computations are typically done using a Bayesian probability combination mechanism. You would use a possibility theory and some logic formalism when there is a logical dependency in sensory interpretation: ‘this datum could be explained by an object here, or by a reflection here and an object there’.

- **Incomplete knowledge**
  The knowledge about the environment is incomplete because of the nature of sensing: when an object is blocking your view, you do not know what is behind it. You may be able to deduce presence of objects by reasoning on a-priori knowledge (‘all objects are square, so I know how to extend this one side I have seen’, or ‘I have already seen all 5 chairs; therefore the area behind this obstacle is free space’).

Uncertainty and incompleteness are different: typically, you can reduce uncertainty by sensing again from the same location; but you cannot reduce the incompleteness in that way. Both are essential to the representation of space in autonomous systems; but both are typically not part of generally
accepted mathematical formalizations of space, so we are on our own in choosing the appropriate representation. We will get back to this issue in chapter 10.

## 7.4 Computer representation of spaces

When considering computer representations of spaces, we need to distinguish discrete spaces and continuous spaces.

### 7.4.1 The graph structure of a discrete space

All representations of discrete spaces will need to contain points, their connections, and (for metric spaces) a notion of distance. In almost all representations, path-distance is used, so that the only distances we need to indicate in the representation are the lengths of the connections – or in other terms, as soon as we want to indicate the distance between two points, we will define a connection between them that has this distance as this length.

We are thus naturally led to a **graph** as a data structure representing the essence of a discrete space. We simply identify a point of the discrete space with a node in the graph, a connection with an edge, and implement an elementary distance as an edge attribute. A metric space, with symmetric distances between the points by the metric properties, would demand an undirected graph with symmetrical edges, and symmetrical distance attributes. However, since the spaces we want to discretize will often be of a higher level of abstraction than actual physical space, **directed graphs** are permitted. They are used if we want to implement certain rules about reality, such as traffic regulations or some simple physics. Two examples of their use:

- **a one-way street**: it is true that physical space permits motions in all directions, but the rules of traffic may not allow this. In a city with one-way streets, the distance between the beginning and the end of the street is 1, but between the end and the beginning it can be 3, since you would have to go around the block.

- **a hill**: we can represent a hill on a 2-dimensional grid of coordinates. The effect of the hill is to make upwards movements slower, and downward motions faster. In this case, the connections between points can be symmetrical, but their distances (in terms of 'time required') are not; we therefore do not, strictly speaking, have a metric.

We note that in a graph representing a discrete space, there is a direct correspondence between node and point, and between edge and connection.
7.4.2 Digitized continuous spaces

We have remarked before that discrete metric spaces have a similar structure to Riemannian metric spaces; and now that we have found a computer representation for one, we may want to use it for the other.

This is commonly done: Riemannian spaces are represented by graphs. But we should realize that the correspondence between real points and nodes in the graph, and between real nodes and points in the graph, is more complicated than in the case of discrete spaces. It requires our full attention in every case we represent a space, or the conclusions we draw on the basis of our representation will have no meaning to the real state of the world.

First, a node in a graph representing a continuous space denotes a set of points of the space; we will call this the region of sensitivity of the node (a term obviously derived from sensing!). These regions need not be chosen to partition the original space, such that each real point contributes to one and only one node; often, it is sensible to make them overlap. You may view each node as a virtual sensor, which may be polled to give the latest estimate on the presence of objects or free space in the region it represents. When viewed that way, you can apply sampling theory to choose the regions of sensitivity for given environmental characteristics such that the discrete representation can be interpreted to within a known accuracy. The issue of representing the presence of objects in the world using a discrete graph is similar to that of choosing the proper resolution and sensor to record an image of a binary scene (and there, too, you should not identify the region of sensitivity of a pixel with the square of points that are closer to the location of its center than to any of its neighbors; indeed, the region of sensitivity is completely determined by the optic characteristics of the camera – and therefore usually shaped like a Bessel function).

Since a node represents a set of points, its contents should represent some measure of the presence of objects and free space in the region of sensitivity; for instance in a known world you could take the object density, and in an unknown world you take an estimated object density, estimated using the sensory data and/or a-priori knowledge. However, along that same density axis, you cannot also denote incompleteness of knowledge; here an odds likelihood measure appears to be more appropriate (see chapter 10).

A second issue in the discrete representation is that the edges of the graph represent connections between sets of real points. You have to specify whether an edge means that all points in both regions of connectivity of the two nodes are involved, or just that there is a connection, or what (both are done in literature). And a similar problem occurs in the assignment of a length to the discrete connection: since there are many real paths, there are many distances; are you going to take the shortest, the longest, the average, or what? The appropriate choice depends on your problem. If you would subsequently use the discrete graph representation for path planning, you
might be interested in a good path which when executed will take the time computed, or less; so in that case you would use a worst-case estimate.

Different graph representation methods of continuous spaces vary in the way they choose the regions of sensitivity, and how they tabulate uncertainty and incomplete knowledge. The determination of a reasonable length measure is usually only secondary. We give two examples, differing in the choice of nodes:

- **Occupancy grids**
  
  An occupancy grid is a network of spatial nodes, of which the layout and the connections are chosen to be that of free space. So, for instance, nodes of the network are associated with locations in the continuous space, the connections with actual possible motions in the absence of obstacles, and distances which are some approximation to the actual distances in the continuous pre-image. As such, this representation is a substrate for representing the state of the world. You would commonly implement this as an array, like you would a matrix; but the connectivity and distance between matrix elements show that its abstract structure is in fact that of a graph.

  On this substrate, you can define a function which you use to denote your knowledge about the presence of objects. For instance, you might denote at each node the *density* of real object points over the region of sensitivity associated with the discrete nodes.

  The density function may be considered as a probability density function: it gives the probability of encountering an object if a real point is placed at a location associated with the discrete nodes.

  A typical application of occupancy grids is as a way to tabulate sensory data: you interpret the sensory events geometrically, and store their interpretation at the nodes representing the corresponding locations. If the sensor is sonar-like, the data you retrieve give an indication of how far the sonar beam got before it encountered an object: so one sensory datum gives an indication of both free space and an object boundary. In chapter 10, we will treat various ways of logging such information at the nodes; here we only state that the data structure for all such spaces is geometrically similar.

- **Quadtrees and octtrees**

  If your space is largely known, an occupancy grid may be less useful. Typically, the sampling density would have to be chosen such that the objects are ‘imaged’ to sufficient detail; but that mostly leads to a characterization where large areas of totally free space, or totally object, still require many nodes. In such known spaces, it makes sense to choose the nodes and their regions of sensitivity more efficiently.
7.4. **COMPUTER REPRESENTATION OF SPACES**

A common way of doing so is, in 2-D, a *quadtree*; and the basic idea easily generalizes to more dimensions, as *octree* in 3D, etcetera. In this approach, a block of \(n\)-dimensional space is split through the middle in \(2^n\) pieces, recursively. Each piece is represented by a node in a tree, with as a parent the block whose split generated it, and as possible \(2^n\) children the blocks in a subsequent split. At each node, it is indicated whether the block of space it represents is **EMPTY** (no objects), **FULL** (all object), or **MIXED** (both object and free space). Only the **MIXED** nodes are split again. Doing this starting from the whole space, splitting down to a desired accuracy produces a tree, see figure 7.8. In that figure, the tree structure is given by the splits at the various level; not depicted is the graph structure corresponding to the connectivity. (Q: For an octree of a few levels, draw the connectivity graph. Find a clever way of labelling nodes such that you can quickly determine whether two nodes represent neighboring regions, just from their codes. How would you define their distance?)

![Diagram](image)

**Figure 7.8**: A scene (a), part of its octree representation (b), and the actual world description to level 5 of the octree.

Although octrees are rather efficient compared to occupancy grids, they have some disadvantages:

- they can change abruptly under small but simple motions of the objects, such as translations or rotations; so similar scenes may
have very different representations

- although it is not hard to come up with distances, it is hard
to find ones that are representative of the real distances in the
original space;

- the accuracy tends to vary quite a bit throughout the representa-
tion (in general, it is better close to object boundaries – this
may be a desirable effect!)

- even when the objects are known exactly, MIXED nodes occur as an
artifact of this representation. When knowledge is uncertain or
incomplete, this can also only be expressed by the MIXED nature
of a node. Thus 3 different concepts are linked in this representa-
tion, which is bad practice.

When an octree is used to represent a space, you are usually able to see
its effect in the behavior of the autonomous system that uses them, say
for path planning. Since we would generally prefer that behavior to be
independent of the format of the internal representation, octrees are
not very elegant for this purpose. They are a technique developed for
coding pictures (and spaces) efficiently, in just a few bits, for efficient
storage and transmission; and maybe that is all we should use them
for.

Many refinements have been made to the octree principle, mainly
involving ways of splitting, and keeping more descriptive indicators
at the nodes. For instance, you might indicate the ‘wildness’ of a
MIXED area at its node (characterized, say, by its spatial correlation
coefficient), and prefer to split areas of greater wildness because you
will need a finer discretization there. And you might find the connected
components in an image and define the spatial splitting to coincide
with the values of their minimum and maximum extent. If you are
intent on using octtrees, we recommend that you survey the literature;
a splitting method that serves you purpose is likely to exist, possibly
even as a program.

Quadtrees are also used in languages for spatial queries; so they are
designed more to answer individual questions like: ‘what is my knowl-
dge about the occupation of this particular real region?’ rather than
to code for a spatial structure. If your spatial problem can be formu-
lated in terms of such queries, you can use quadtrees, or other spatial
indexing structures, see e.g. [4].

To summarize: even though both discrete and continuous spaces are
represented by graphs, these graphs have a different interpretation in the
two cases. And often this difference in semantics necessitates a different
syntax, for instance in the manner in which the distance between two nodes
is computed from the elementary lengths of edges connecting the nodes.
7.5 Modelling

We have seen in chapter 2, that an autonomous system resides in the real world in a kind of closed loop. It perceives this world through its sensors and interacts with the real world through its actuators. The required interaction is based on a model of the real world: the world model. This model accumulates the knowledge we have acquired so far. Based on the perception of the real world we build and adapt this world model. An example of a world model is a map of the environment in which the walls and obstacles are represented. The map could be obtained through ultrasonic sensors, in which case distances in certain directions are used to update the map. But if the observation is obtained from a camera, also colours and textures could be represented in the map of the environment. Specific walls in the environment could be found back based upon this information. So, the world model contains only those aspects of the real world which can be perceived by the sensors and represents that kind of information needed to react or to reason about the actions to be performed.

However, we also need to have an idea of how the actions will influence the world. This is not always trivial, and in those cases we need a model of the system behavior: the actuator model. If we can directly actuate a two-link robot arm in the task space, i.e. we can specify the position of the end effector in Cartesian world coordinates \((x, y)\), we don’t need an actuator model. When the actuation takes place in the configuration space, we have to specify the position of the arm in the values of the joints \((\theta_1, \theta_2)\). We have to know what the position will be in world coordinates, and need a model which describes this relation.

In figure 7.9 the interaction between the autonomous system and the real world is represented in a diagram, in which the world model describes the real world based on the information obtained from the (virtual) sensors. The desired changes in the real world are based upon this model. The actuator model describes what actions are needed to make this changes. We can only make those changes to which the actuators are capable. We could also speak of a sensor model which describes the perception of certain features in the real world. An ultrasonic sensor, for example, generates a sound pulse and uses a sensor model to convert the delay in emission and reception into a distance in a certain direction. We can only perceive those features of the environment which can be sensed by the sensors.

In chapter 2 we saw that models exists on all different levels. Each level has its own abstraction level, based on its virtual sensors and actuators.

A world model enables us to fuse the information of different types of sensors and to combine sensor measurements from successive time intervals. A model gives the possibility to fuse the measurements directly in the context of the interpretation sensor data. The combination of sensors of different types has the following advantages:
1. Redundant information. Redundant information is obtained when each sensor perceives the same features in the environment. By combining the sensor information, the overall uncertainty of the perceived features can be reduced. Redundant information is especially important in case of a sensor error or failure. This is the case for sensors of both one and several type. When all but one sensor obtain consistent information, a malfunctioning sensor can be detected.

2. Complementary information. Complementary information allows the extraction of features that are impossible to perceive using the information from one individual sensor, or even one sensor type. One vision sensor does not produce a complete depth map, while a stereo vision system does. Neither of them detects a glass wall, in which case the addition of a ultrasonic sensor is useful.

3. Time duration. Time savings in obtaining information can be achieved by using multiple sensors rather than a single sensor. This can be achieved either because of the actual speed of operation of each sensor or by exploiting the possible parallelism inherent in the combination of the information. This is even the case when only one type of sensor is used.

4. Cost of information. In a number of cases, sensor information can be obtained at a lower cost by using multiple sensors. This is because important savings can be obtained by using algorithms which obtain the information cheaply, or only extract the required information.

A model will not be an exact description of the reality, it is not complete. It approximates the reality only up to a certain level and accuracy,
incorporating only those aspects needed for the required type of interaction. Whether or not the model approximates reality sufficiently is not always possible to say beforehand. Reality is complex and many unforseen details can influence the expected behavior. Simulation of the model gives the possibility to investigate the accuracy by comparing the calculated behavior with the real results. Simulation enables us also to test whether the designed strategies for autonomy work. It is much easier to test a proposed system in simulation than in reality.

Model elements
A model should contain the following elements:

- input variables
- output variables
- set parameters
- (hidden) internal parameters and variables
- specification how to produce output given input, and set parameters; this specification often involves hidden parameters and variables.
- an indication of the accuracy of the model.

Let us first consider a simple example. We will consider the two-link robot arm of the previous section. This robot arm consists of two links with lengths $\ell_1$ and $\ell_2$ and is sketched in figure 7.10. The joint angles of the arm in the down position are $\theta_1$ and $\theta_2$, the angles of the upper position $\theta'_1$ and $\theta'_2$. The angle of the object is denoted by $\theta_o$.

![Figure 7.10: Geometry of two-link robot arm with $\theta_o$ the angle of the object](image)

\[ \alpha = \theta_o - \theta_1 \]
\[ \theta'_1 = \theta_o + \alpha = 2\theta_o - \theta_1 \]
\[ \theta'_2 = -\theta_2 \]
CHAPTER 7. REPRESENTATION AND MODELING

The position of the gripper \((x, y)\) depends upon the joint angles of the links \((\theta_1, \theta_2)\) and the length of the links \(\ell_1\) and \(\ell_2\) and is given by:

\[
x = \ell_1 \cos \theta_1 + \ell_2 \cos(\theta_1 + \theta_2) \\
y = \ell_1 \sin \theta_1 + \ell_2 \sin(\theta_1 + \theta_2).
\] (7.38)

When we have detected the position \((x, y)\) of the object to be grasped, we want to know the values of the joints \((\theta_1, \theta_2)\) to arrive at that position. Therefore we need a model of our robot system which tells us what joint values are required to bring the gripper at the right position in real space. So we need an actuator model of the so-called inverse kinematics of the robot arm. In this case the **input variables** are \((x, y)\) and the **output variables** \((\theta_1, \theta_2)\). The **specification** is in the form of two equations given in 7.39. These equations can be derived from equation 7.38 and this derivation can be found in e.g. [5] (page 108):

\[
\cos \theta_2 = \frac{x^2 + y^2 - \ell_1^2 - \ell_2^2}{2\ell_1 \ell_2} \\
\theta_1 = \tan^{-1}\left(\frac{y}{x}\right) - \tan^{-1}\left(\frac{\ell_2 \sin \theta_2}{\ell_1 + \ell_2 \cos \theta_2}\right).
\] (7.39)

If we know \(\ell_1\) and \(\ell_2\) we can compute \(\cos \theta_2\) corresponding to a given position \((x, y)\). The lengths \(\ell_1\) and \(\ell_2\) are the **internal hidden parameters**. We can calculate \(\cos \theta_2\) from equation 7.39, however, there are two possible values for \(\theta_2\), as \(\cos \theta_2\) has the same value for \(\theta_2\) (down) and \(-\theta_2\) (up). We could see this also in another way: we can grip an object in two configurations with an elbow up or elbow down. Both configurations are sketched in figure 7.10. This choice whether we want to grip with elbow up or elbow down forms a **set parameter**: \(s\) with values +1 or -1. When the space is free, this choice is arbitrary. When there are obstacles present, we could use this set parameters to avoid a collision.

The accuracy of a model is often a difficult question. To investigate that, we should know which details we are discarding and how big their influence on the model is. Simulation is an important tool to give insight in the accuracy, by comparison of the values calculated from the model and the values measured from the real world. In our example we should like to know how close the gripper will come to the object position \((x, y)\), when we send the joint values \((\theta_1, \theta_2)\) to the robot actuator.

Aspects we are discarding are for instance: i) backlash, ii) bending of the links due the weight of a load, iii) error in the (end) joint position due to friction and control bias.

- **Backlash error**: can be largely avoided but using special gears.
- **The bending**: is an very important error if we would like to make arms which are more human like: light and less stiff than the rigid and heavy
industrial robots. An example of such an arm is the Rubbertuator arm from the Bridgestone company. For industrial robot arms this error it can be discarded.

- **The error in joint end position due to control errors and friction** is in general the most important error. For the OSCAR robot this error is for instance around 0.5 degrees in the first 3 joints.

<table>
<thead>
<tr>
<th>input variables</th>
<th>$(x, y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>output variables</td>
<td>$(\theta_1, \theta_2)$</td>
</tr>
<tr>
<td>set parameters</td>
<td>$s$ (+1 or -1)</td>
</tr>
<tr>
<td>hidden internal parameters and variables</td>
<td>$(\ell_1, \ell_2)$</td>
</tr>
<tr>
<td>specification how to produce output given input and set parameters; specification often involves hidden parameters and variables</td>
<td>$\theta_2 = s \cdot \cos^{-1} \left( \frac{x^2 + y^2 - \ell_1^2 - \ell_2^2}{2 \ell_1 \ell_2} \right)$; $\theta_1 = \tan^{-1} \left( \frac{y}{x} \right) - \tan^{-1} \left( \frac{\ell_2 \sin \theta_2}{\ell_1 + \ell_2 \cos \theta_2} \right)$</td>
</tr>
<tr>
<td>indication of the accuracy of the model</td>
<td>OSCAR robot: 0.5 degrees in $\theta_1$ and $\theta_2$</td>
</tr>
</tbody>
</table>

Table 7.1: Inverse kinematics model of a two-link robot arm

In the inverse kinematics example, the model consists of mathematical equations, relating the output values $(\theta_1, \theta_2)$ to the input values $(x, y)$. These equations involve set parameters, which are in a way fixed constants to the model. Important are also the (hidden) internal parameters of the model, which have to be measured or estimated, before we can use the model. In our example the parameters had a physical meaning, the length of the robot links. We could obtain them by measuring the length of the links, or use the values given by the manufacturer. A better way is to find the parameters for the specific robot we are using. We can find them by a calibration procedure, in which the robot makes arbitrary moves. As a result we obtain a set of $N$ measured positions $(x, y)$ corresponding to a set of joint angles $(\theta_1, \theta_2)$. We have to find now those parameter values that minimize the difference between the $N$ outputs of the model (the computed positions) and the measured positions. This is called parameter estimation and will be discussed in the next section in more depth.

Adaptation

When the system changes its behavior during operation, e.g. due to wear and tear, it would be nice if we could apply the calibration procedure continuously. We would like that the model automatically adapts itself to the
changing behavior. Therefore it should be possible to obtain continuously measurements of the input and output variables of the model. Each time we have acquired $N$ measurements we could calculate the optimal parameters over the last $N$ time-intervals, and automatically adapt to changes.

You could ask whether it is possible to adapt directly after each new measurement and not having to wait, until we have assembled $N$ measurements. This can indeed be done, and the technique is called Kalman filtering. It enables us to get the optimal parameters after each measurement, and is discussed in depth in section 7.6.2.

So far the model we discussed consisted of equations of which the parameters had to be estimated. These models apply, when the representation of the input and output variables is numeric, which is typically so on the lower levels of the autonomous system. Such a lower level model may form the virtual sensor for a higher level.

An example is a virtual wall sensor in a free space model. Assume we want to model the free space in which a mobile robot may dwell. The virtual wall sensor should model the position of a wall and fits ultrasonic data to a straight line. The resulting wall parameters from the model form the output of the virtual sensor.

Another example is the road driving case study, where the modelling of the road position can be seen as a virtual sensor which is described in the numeric domain. Modelling free space occurs frequently with mobile robots. In its most simple form, the model only gives the chance that a given cell of some discrete space representation is occupied. In more complex models the occupied space is interpreted and the model represents walls, doors and obstacles. Walls have then a position but also a texture or landmarks on it, so that a specific wall can be retrieved. In general these higher levels become more application specific, and will incorporate non-numeric data. But, they rely in general on numeric models as basic input.

**Explicit and implicit models**

An explicit model is a model, which is based on physics (or chemistry, biophysics etc.). Our knowledge of the world and insight in the kind of interactions lay the basis for the model and as a result the parameters of an explicit model have a physical meaning. In the example we discussed previously the model of the inverse kinematics of the robot arm was an explicit model, it was based on geometry and the parameters had a physical meaning: it were the lengths of the robot links. An implicit model can be seen as a kind of black box. It describes as good as possible the input-output relation and its parameters are simply adjusted in such a way that the difference between model and reality is minimized. The parameters of an implicit model do not have any physical meaning. An example of an implicit model is a feedforward neural network.
In our example we could also use a feedforward neural network to model the inverse kinematics. The feedforward has 2 inputs \((x \text{ and } y)\) and 2 outputs \((\theta_1 \text{ and } \theta_2)\), and has a hidden layer consisting of \(K\) hidden nodes. We thus obtain:

\[
\theta_1 = \sum_{j=1}^{K} (w_{j\theta_1} \phi(w_{xj}x + w_{yj}y + w_{0j}) + w_{0\theta_1})
\]

\[
\theta_2 = \sum_{j=1}^{K} (w_{j\theta_2} \phi(w_{xj}x + w_{yj}y + w_{0j}) + w_{0\theta_2}).
\]

(7.40)

with \(K\) the number of nodes and \(w_{ij}\) the weighting factors. The configuration of the network is shown in figure 7.11.

![Feedforward Neural Network](image)

**Figure 7.11: A Feedforward Neural Network**

In such a network the linearly weighted inputs \(x\) and \(y\) form the input to non-linear functions \(\phi\) (e.g. sigmoidal functions). These functions form the hidden nodes of the network. The linear weighted nodes approximate the required function.

It has been shown ([6], [9]) that feedforward neural networks can attain an arbitrary accuracy of approximation if sufficiently many hidden nodes are available.

Also in an implicit model the weights have to be optimised to minimize the difference between the model and the measurements. The same minimization techniques as applied for explicit models, can be applied here for implicit models. The real difference is the origin of the model and the meaning of the parameters, based on an understanding of the real world or not. Even the popular technique of back-propagation in feedforward neural
networks optimization, in which the weights $w_{ij}$ are updated based upon the successive samples, is related to the Kalman filtering discussed in section 7.6.2. However, there is an important observation to be made. In an explicit model we incorporate knowledge about the real world. This knowledge doesn’t have to be estimated. This results in general in less parameters in an explicit model than in an implicit model. For the two link robot example the explicit model has 2 hidden parameters $\ell_1$ and $\ell_2$. The implicit model has for each node 5 parameters $w_{ij}$. A typical value for such a network is 5 nodes, resulting in 27 parameters to estimated! In general, when we have more parameters, we need more examples for estimation to arrive at the same accuracy of the model.

### 7.6 Explicit models

#### 7.6.1 Parameter estimation techniques

The basic approach in parameter estimation is to minimize the difference between the output data of the model and the measured data from the real world. This is realized by optimizing the parameter values so that the measured data fits as good as possible the model.

There are some important aspects to be noticed. Measured data is not exact. The data is subject to measurement errors called noise. Thus, typical data never will exactly fit the model, even when the model is completely correct. To determine whether the model is appropriate, we need a goodness-of-fit criterion which can be tested against some statistical standard, based upon the measurement errors.

To be useful, a fitting procedure should provide not only (i) the parameters but (ii) also error-estimates of the parameters and (iii) a statistical goodness of fit measure. This last measure indicates whether the model does describe the data well, or not.

Suppose that we are fitting $N$ data points $(x_i, y_i)$ to a model which has $M$ adjustable parameters. The model gives the relationship between the input values $x$ and the output values $y$:

$$y(x) = y(x; a_1, a_2, \ldots, a_M)$$  \hspace{1cm} (7.41)

in which the dependency upon the parameters $a_1, a_2, \ldots, a_M$ is explicitly shown. When we minimize (with respect to $a_1, a_2, \ldots, a_M$) the quadratic error between the data points $y_i$ and the model values $y(x_i)$, we obtain a least-squares fit:

$$E = \sum_{i=1}^{N} [y_i - y(x_i; a_1, a_2, \ldots, a_M)]^2.$$  \hspace{1cm} (7.42)

In general this means that starting at some initial values of the parameters $a_1, a_2, \ldots, a_M$, we have to iteratively adapt them to lower $E$. The change in
7.6. **EXPLICIT MODELS**

\[ a_1, a_2, \ldots, a_M, \] giving the largest decrease in \( E \) (steepest decent) can be obtained from \( \frac{\partial E}{\partial a_i} \). But there exist a wealth of (more intelligent) minimization procedures, see [10]. The error landscape as a function of the parameters can have local minima, so that we have to be careful with the minimization procedures not to get stuck there.

Life is much more easy, when the model is linear. In that case we can directly solve equation 7.42. Setting the \( \frac{\partial E}{\partial a_i} \) to zero give \( M \) linear equations from which we can solve the parameters \( a_i \). We will show this in two examples.

**Example 1: Fit \( N \) sensor measurements to a straight line.** This is the problem when we like to fit a wall to sensor data, in which case we want to know the position of the wall, so the parameters \( a \) and \( b \):

\[ y = ax + b \quad (a = \tan \alpha). \quad (7.43) \]

This problem is illustrated in figure 7.12. The straight line model predicts the relationship between the input of the model (the \( x_i \) coordinates of the \( N \) data points) and the output of the model (the corresponding computed \( y \) coordinates, denoted by \( y(x_i) \)).

![Image of a straight line with parameters](image)

Figure 7.12: Estimation of the line parameters for a set measured data points

We want to compare the measured values \( y_i \) with the computed values \( y(x_i) = ax_i + b \). The error \( E \) is given by:

\[ E = \sum_{i=1}^{N} [y_i - y(x_i)]^2 = \sum_{i=1}^{N} [y_i - ax_i - b]^2. \quad (7.44) \]

In this case we can calculate the optimal parameter values analytically. \( E \) is minimal for those values of the parameters \( a \) and \( b \) for which:

\[ \frac{\partial E}{\partial a} = 0 \quad \text{and} \quad \frac{\partial E}{\partial b} = 0. \quad (7.45) \]
It is not difficult to compute the partial derivatives with respect to \( a \) and \( b \):

\[
\frac{\partial E}{\partial a} = -2 \sum_{i=1}^{N} x_i(y_i - ax_i - b) = 0
\]

\[
\frac{\partial E}{\partial b} = -2 \sum_{i=1}^{N} (y_i - ax_i - b) = 0.
\] (7.46)

If we define the following sums:

\[
S_x = \sum_{i=1}^{N} x_i \quad S_y = \sum_{i=1}^{N} y_i \quad S_{xx} = \sum_{i=1}^{N} x_i^2 \quad S_{xy} = \sum_{i=1}^{N} x_i y_i,
\] (7.47)

we get two linear equation in \( a \) and \( b \) for which we have to solve \( a \) and \( b \):

\[
aS_{xx} + bS_x = S_{xy}
\]

\[
aS_x + bN = S_y.
\] (7.48)

From these equations the optimal coefficients \( a \) and \( b \) can be calculated:

\[
a = \frac{N.S_{xy} - S_x S_y}{N.S_{xx} - (S_x)^2}
\]

\[
b = \frac{S_{xx}S_y - S_x S_{xy}}{N.S_{xx} - (S_x)^2}.
\] (7.49)

As was stated before, the computation of the optimal parameters is only the first step of the solution. Now we have to find the error-estimates of the found parameters. We assume that the uncertainty \( \sigma_i \) associated to each measurement \( y_i \) is known, and that the \( x_i \)'s are known exactly. The measurements errors in the data introduce uncertainties in the estimates of \( a \) and \( b \). If we have a function \( f(y_1, y_2, \ldots, y_N) \), with \( \sigma_i^2 \) the variance of \( y_i \), we can propagate the errors in the measurements to the error in the value of the function in the following way:

\[
\sigma_f^2 = \sum_{i=1}^{N} \sigma_i^2 \left( \frac{\partial f}{\partial y_i} \right)^2.
\] (7.50)

We can compute the derivatives \( \frac{\partial a}{\partial y_i} \) and \( \frac{\partial b}{\partial y_i} \) from the found solutions:

\[
\frac{\partial a}{\partial y_i} = \frac{N.x_i - S_x}{N.S_{xx} - (S_x)^2}
\]

\[
\frac{\partial b}{\partial y_i} = \frac{S_{xx} - S_x x_i}{N.S_{xx} - (S_x)^2}.
\] (7.51)
7.6. **EXPLICIT MODELS**

If we sum over \(i\) and we assume that \(\sigma_i\) are the same for all the measurements, we get:

\[
\sigma_a^2 = \sigma^2 \sum_{i=1}^N \left( \frac{\partial a}{\partial y_i} \right)^2 = \sigma^2 \frac{N}{N S_{xx} - (S_x)^2}
\]

\[
\sigma_b^2 = \sigma^2 \sum_{i=1}^N \left( \frac{\partial b}{\partial y_i} \right)^2 = \sigma^2 \frac{S_{xx}}{N S_{xx} - (S_x)^2}.
\]  \hspace{1cm} (7.52)

These are the variances of the estimates \(a\) and \(b\) respectively.

**Example 2:** Fit \(N\) sensor measurements to the inverse kinematics model. The parameters to be estimated are the lengths \(\ell_1\) and \(\ell_2\). These can be estimated by having the robot make arbitrary moves with joint values \((\theta_{1i}, \theta_{2i})\) and measure the position \((x_i, y_i)\) of the gripper (e.g. with a video camera) for each move. In this way we obtain a set of \(N\) (measured) data points \((x_i, y_i)\) for joint values \((\theta_{1i}, \theta_{2i})\).

Now we have to minimize the distance \(d_i\) between the measured \((x_i, y_i)\) and the model position \((x(\theta_{1i}, \theta_{2i}), y(\theta_{1i}, \theta_{2i}))\) given in equation 7.38:

\[
d_i^2 = (x_i - x(\theta_{1i}, \theta_{2i}))^2 + (y_i - y(\theta_{1i}, \theta_{2i}))^2 = \{x_i - \ell_1 \cos \theta_{1i} - \ell_2 \cos(\theta_{1i} + \theta_{2i})\}^2 + \{y_i - \ell_1 \sin \theta_{1i} - \ell_2 \sin(\theta_{1i} + \theta_{2i})\}^2.
\]  \hspace{1cm} (7.53)

The total error \(E\) is obtained by summing up all \(N\) values of \(d_i^2\):

\[
E = \sum_{i=1}^N d_i^2. \hspace{1cm} (7.54)
\]

The minimum of \(E\) is obtained when the derivatives of \(E\) with respect to the parameters \(\ell_1\) and \(\ell_2\) are 0, so:

\[
\frac{\partial E}{\partial \ell_1} = 0 \quad \text{and} \quad \frac{\partial E}{\partial \ell_2} = 0. \hspace{1cm} (7.55)
\]

We first compute the parts of the sums:

\[
\frac{\partial E_i}{\partial \ell_1} =
2\{x_i - \ell_1 \cos \theta_{1i} - \ell_2 \cos(\theta_{1i} + \theta_{2i})\} \cdot -\cos \theta_{1i} +
2\{y_i - \ell_1 \sin \theta_{1i} - \ell_2 \sin(\theta_{1i} + \theta_{2i})\} \cdot -\sin \theta_{1i}
= 2\{\ell_1 + \ell_2 (\sin \theta_{1i} \sin(\theta_{1i} + \theta_{2i}) + \cos \theta_{1i} \cos(\theta_{1i} + \theta_{2i}))
- x_i \cos \theta_{1i} - y_i \sin \theta_{1i}\}
\]
\[
\frac{\partial E_i}{\partial \ell_2} = 2\{x_i - \ell_1 \cos \theta_{1i} - \ell_2 \cos(\theta_{1i} + \theta_{2i})\} - \cos(\theta_{1i} + \theta_{2i}) + 2\{y_i - \ell_1 \sin \theta_{1i} - \ell_2 \sin(\theta_{1i} + \theta_{2i})\} - \sin \theta_{1i} + \theta_{2i}
\]
\[
= 2\{\ell_1 (\sin \theta_{1i} \sin(\theta_{1i} + \theta_{2i}) + \cos \theta_{1i} \cos(\theta_{1i} + \theta_{2i})) + \ell_2 - x_i \cos \theta_{1i} + \theta_{2i}) - y_i \sin(\theta_{1i} + \theta_{2i})\}. \tag{7.56}
\]

From the formula for the cosine of the sum of two angles we can derive that:
\[
\sin \theta_{1i} \sin(\theta_{1i} + \theta_{2i}) + \cos \theta_{1i} \cos(\theta_{1i} + \theta_{2i}) = \cos \theta_{1i} \cos \theta_{2i}. \tag{7.57}
\]

We can compute the sums of the parts and equate them to zero:
\[
\sum_{i=1}^{N} \frac{\partial E_i}{\partial \ell_1} = 2\{\ell_1 \sum_{i=1}^{N} 1 + \ell_2 \sum_{i=1}^{N} \cos \theta_{2i} - \sum_{i=1}^{N} x_i \cos \theta_{1i} - \sum_{i=1}^{N} y_i \sin \theta_{1i}\} = 0
\]
\[
\sum_{i=1}^{N} \frac{\partial E_i}{\partial \ell_2} = 2\{\ell_1 \sum_{i=1}^{N} \cos \theta_{2i} + \ell_2 \sum_{i=1}^{N} 1 - \sum_{i=1}^{N} x_i \cos \theta_{1i} + \theta_{2i}) - \sum_{i=1}^{N} y_i \sin(\theta_{1i} + \theta_{2i})\} = 0. \tag{7.58}
\]

If we further substitute:
\[
S_1 = \sum_{i=1}^{N} x_i \cos \theta_{1i} + \sum_{i=1}^{N} y_i \sin \theta_{1i}
\]
\[
S_2 = \sum_{i=1}^{N} x_i \cos(\theta_{1i} + \theta_{2i}) + \sum_{i=1}^{N} y_i \sin(\theta_{1i} + \theta_{2i})
\]
\[
P = \sum_{i=1}^{N} \cos \theta_{2i} \tag{7.59}
\]

we get:
\[
N \ell_1 + P \ell_2 = S_1
\]
\[
P \ell_1 + N \ell_2 = S_2. \tag{7.60}
\]

The solution of these equations can be calculated and gives us the best-fit parameters \(\ell_1\) and \(\ell_2\):
\[
\ell_1 = \frac{NS_1 - PS_2}{N^2 - P^2}
\]
\[
\ell_2 = \frac{NS_2 - PS_1}{N^2 - P^2}. \tag{7.61}
\]
7.6.  **EXPLICIT MODELS**

Model validation

We would like to obtain a measure of the goodness-of-fit of the model to the measurements. Could the errors originate from the measurement errors, or does the model not fit the data? When we assume that the measurement errors in the \( y_i \) values are independent of each other and normally distributed around the hypothetical true model values \( y(x_i) \), then we can use a statistical test: the \( \chi^2 \)-test. When we divide the minimum value of \( E \) by the variance \( \sigma^2 \) of the measurement errors, we obtain a variable called \( \chi^2 \):

\[
\chi^2 = \frac{E_{\text{min}}}{\sigma^2}. \tag{7.62}
\]

So, if the model is correct, the error should be distributed according to a \( \chi^2 \)-distribution with \( N-M \) degrees of freedom, in which \( N \) is the number of samples and \( M \) the number of parameters. The value of \( \chi^2 \) obtained for the best-fit parameters can be looked up in a table, which gives us the probability that this value occurred caused by the measurement errors. This probability gives a quantitative measure for goodness-of-fit of the model. If this probability is small for some particular data set, then the apparent discrepancies are unlikely to have occurred by chance, so the model is wrong (or the variance in the measured values is larger than expected).

**\( \chi^2 \) example**  Assume for instance in example 1 that we fit a straight line to 20 measured points. So, \( N = 20 \) and as there are 2 parameters \( M = 2 \). After fitting the data the error \( E_{\text{min}} = 241.3 \) and the standard deviation is 2.7. We obtain:

\[
N - M = 18
\]

\[
\chi^2 = \frac{E_{\text{min}}}{\sigma^2} = 33.1.
\]

![Figure 7.13: \( \chi^2 \) distribution for 18 degrees of freedom.](image)
In table 7.2 the 95% $\chi^2$ values are given. When a $\chi^2$ value is smaller than the value listed in the table, it falls within the 95% probability interval that this value occurred by chance. And when the $\chi^2$ value is greater it is in the 5% probability interval. In the last case we have to reject the model.

In figure 7.13 the distribution of $\chi^2$ is given with 18 degrees of freedom. The table value 28.9 divides the area under the curve in a 95% probability interval to the left, and a 5% probability interval to the right. We found a value 33.1, which lies in the 5% probability interval. This means we have to face the fact that the model is not correct.

<table>
<thead>
<tr>
<th>$N - M$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\chi^2$</td>
<td>3.84</td>
<td>5.99</td>
<td>7.81</td>
<td>9.49</td>
<td>11.1</td>
<td>12.6</td>
<td>14.1</td>
<td>15.5</td>
<td>16.9</td>
<td>18.3</td>
</tr>
<tr>
<td>$N - M$</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>$\chi^2$</td>
<td>19.7</td>
<td>21.0</td>
<td>22.4</td>
<td>23.7</td>
<td>25.0</td>
<td>26.3</td>
<td>27.6</td>
<td>28.9</td>
<td>30.1</td>
<td>31.4</td>
</tr>
<tr>
<td>$N - M$</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>$\chi^2$</td>
<td>32.7</td>
<td>33.9</td>
<td>35.2</td>
<td>36.4</td>
<td>37.7</td>
<td>38.9</td>
<td>40.1</td>
<td>41.3</td>
<td>42.6</td>
<td>43.8</td>
</tr>
</tbody>
</table>

Table 7.2: 95% $\chi^2$ values as function of the degrees of freedom

**Outliers** Sometimes completely wrong measurements, so called outliers are present. They may ruin the accuracy of the model. Outliers can be suppressed by introducing weights $w_i$ in the error contributions of the different measurements in $E$ (see equation 7.63). By iteratively applying the estimation procedure and letting the weights decrease when they are too far from the model, we can make the parameter estimation robust to outliers. A typical weight as function of the distance to the model is given in figure 7.14. When the measurements deviate less than $2\sigma$ from the model, all weights $w$ are equal. In the region $2\sigma$ to $3\sigma$ the weight goes to zero. The weight of measurements deviating more than $3\sigma$ is 0.

$$E = \sum_{i=1}^{N} w_i [y_i - y(x_i; a_1, a_2, \ldots, a_M)]^2. \quad (7.63)$$

We refer the interested reader to Press et al. [10] for more detailed information and further reading on both linear and non-linear models.

### 7.6.2 Recursive parameter estimation

**Introduction**

In the previous section we have seen how measurements can be fit to a model by minimizing the difference between the output of the model and the measured data from the real world. Now we show how we can estimate system parameters which are changing. This is done in a recursive way by
computing new estimates of the parameters each time a new measurement is taken. Recursive means that the new estimates are given as a function of the old estimates. Because we now deal with a changing system, we use a dynamic model the system which tells us how the system changes during time. In the examples we show how sensor measurements can be combined to the dynamic model to obtain an estimate of the system parameters.

**Static Example**

To see how 2 sensors can be combined to obtain a more accurate estimate, a simple example will now be given. Suppose we want to determine the position of a vehicle combining two independent measurements given by sensor 1 and sensor 2. These measurements are somewhat uncertain. For the sake of simplicity, consider a one-dimensional location. Suppose we obtain measurement \( y_1 \) and measurement \( y_2 \) from sensor 1 resp. sensor 2 at the same time with variances \( \sigma_1^2 \) and \( \sigma_2^2 \). (The variance says something about the precision of the measurement: if \( \sigma_2^2 \) is smaller than \( \sigma_1^2 \), one has to trust measurement 2 more than measurement 1). We want to combine these measurements to form a weighted mean \( x \) with minimum variance and hence maximum probability. The general form of the weighted mean of \( y_1 \) and \( y_2 \) with weighting factor \( w \) is:

\[
x = (1 - w)y_1 + wy_2.
\]  

(7.64)

What is the required weighting factor \( w \) which gives us an estimate with minimum variance?

The expectation of \( x \) is:

\[
E(x) = (1 - w)E(y_1) + wE(y_2).
\]  

(7.65)
The variance of $x$ is:

$$
\sigma^2 = E \left\{ (x - E(x))^2 \right\} = 
E \left\{ ((1 - w)y_1 + wy_2 - (1 - w)E(y_1) - wE(y_2))^2 \right\} = 
(1 - w)^2 E \left\{ (y_1 - E(y_1))^2 \right\} + w^2 E \left\{ (y_2 - E(y_2))^2 \right\} = 
(1 - w)^2 \sigma_1^2 + w^2 \sigma_2^2.
$$

(7.66)

We now determine $w$ for which $\sigma^2$ is minimal, by taking the first derivative with respect to $w$ and equate this to zero:

$$
\frac{d\sigma^2}{dw} = -2(1 - w)\sigma_1^2 + 2w\sigma_2^2 = 0 \Rightarrow \hat{w} = \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}.
$$

We denote the optimal weighting factor by $\hat{w}$. This means the minimum variance estimate $\hat{x}$ is equal to:

$$
\hat{x} = \frac{\sigma_2^2 y_1 + \sigma_1^2 y_2}{\sigma_1^2 + \sigma_2^2}
$$

(7.67)

and for the (minimum) variance the following equation holds:

$$
\frac{1}{\sigma^2} = \frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}.
$$

(7.68)

This means the uncertainty in the estimate of the position has been decreased by combining two independent measurements. So, this is a linear (in $y_1$ and $y_2$) estimate whose variance is less than that of any other linear estimate. Let's consider the form of $\hat{x}$:

1. If $\sigma_1$ were equal to $\sigma_2$, which is to say we think the measurements are of equal precision, the equation says the optimal estimate of the position is simply the average of the two measurements, as would be expected.

2. On the other hand, if $\sigma_1$ were larger than $\sigma_2$, which is to say that the uncertainty involved in measurement $y_1$ is larger than that of measurement $y_2$, then the equation weights $y_2$ more heavily than $y_1$.

3. Further, the variance of the estimate $\hat{x}$ is less than $\sigma_1^2$, even if $\sigma_2^2$ is very large: even poor quality data provide some information, and should thus increase the precision of the filter output.

We thus found the following linear minimum variance estimate:

$$
\hat{x} = (1 - \hat{w}) y_1 + \hat{w} y_2.
$$

(7.69)
7.6. **EXPLICIT MODELS**

If we regard \( y_2 \) as a measurement used to correct the updated estimate \( y_1 \) and we rewrite the previous equation, we have:

\[
\hat{x} = y_1 + \phi(y_2 - y_1). \tag{7.70}
\]

Then, this shows more clearly that the estimate \( y_1 \) is corrected by adding a weighted difference.

**From static to dynamic estimation**

In the example we combined to independent measurements to obtain an estimate with more accuracy than each of the measurements alone. The measurements were obtained at the same time. But now suppose, we travel before taking another measurement of the position. So, suppose we have:

1. - measurement \( y_1 \) at moment \( t_1 \)
2. - measurement \( y_2 \) at moment \( t_2 \)
3. - best prediction of the position based on measurement \( y_1 : \hat{x}_1 \).

What is the estimate of the position \( \hat{x}_2 \) at moment \( t_2 \)? The Kalman filter gives an answer to this question:

\[
\hat{x}_2 = \hat{x}_1 + K_2(y_2 - \hat{x}_1) \tag{7.71}
\]

with \( K_2 \) the optimal weighting factor at moment \( t_2 \). The equation says that the optimal estimate \( \hat{x}_2 \) at moment \( t_2 \) is equal to the best prediction of its value \( \hat{x}_1 \) before \( y_2 \) is taken, plus a correction term of an optimal weighting value times the difference between \( y_2 \) and the best prediction of its value \( \hat{x}_1 \) before it is actually taken.

It is worthwhile to understand this **predictor-corrector** structure of the filter. Based on all previous information a prediction of the value that the desired variables and measurements will have at the next measurement time is made. Then, when the next measurement is taken, the difference between it and its predicted value is used to "correct" the prediction of the desired variables.

The Kalman filter is a recursive algorithm, which means that it uses the results of the previous step to aid in obtaining the desired result of the current step. So each new estimate is a blend of the old estimate and current measurement. So, the Kalman filter gives a solution to the dynamic estimation problem, as for example the position of a moving vehicle. For this we need a model of the motion of the vehicle. Suppose the following elements are available:

- a dynamic or evolution model for the behavior of the system
• a measurement model for the relation between the state of the system and the measurements

• a description of the noise.

Suppose further that we can extend previous equations to the vector case, then we can apply the Kalman filter. Before continuing with the Kalman filter, we will first describe the system models which are used to obtain an optimal estimate of a system. Then, we will describe how noise components can be added to the equations, which are then called stochastic equations.

**Linear discrete systems**

The dynamics of a system (the behavior of the system) can be described by equations which represent the relation between the **input**, the **output** and the **state** of the system. The input of the system is the control input from chapter 1 which represents the external sources acting on the system. For example, the throttle input acts on the system to produce a change of the system parameters. The output represents the observable behavior of the system. The state of the system represents the selected quantities we are interested in. The basic property of a dynamic system is that its behavior at any instant depends not only on the variables acting on it at that very instant, but also on the variables having acted on it in the past.

We express input, state and output by vectors: the **input vector** is usually denoted by vector $u$, the **state vector** by $x$, and the **output vector** by $y$. If the input, state and output vectors are defined for each time $t$ of some interval we speak of **continuous-time** systems, and the system equations are differential equations. If the input, state and output vector are defined only at discrete instants $t_k$ ($k = 0, 1, 2 \ldots$) we speak of **discrete-time systems**, and the dynamics of the system is described by difference equations. Of course, linear discrete-time systems are represented by linear difference equations.

The dynamic behavior of linear discrete-time systems can be described by the following difference equations:

$$
\begin{align*}
x_{k+1} &= A_k x_k + B_k u_k & \text{state equation} \\
y_k &= C_k x_k & \text{measurement equation,} \quad (7.72)
\end{align*}
$$

where

1. $k = 0, 1, 2 \ldots$,

2. $A_k$ is an $n$ by $n$ matrix which describes the evolution of the system,

3. $B_k$ is an $n$ by $m$ matrix which describes the influence of the input on the system,
4. $C_k$ ($p$ by $n$) gives the relation between the state and the output,

5. state vector $x_k \in \mathbb{R}^n$, output vector $y_k \in \mathbb{R}^p$ and input vector $u_k \in \mathbb{R}^m$.

In the state equation the next state is defined as a linear function of the previous state and the input. In the measurement equation the measurements $y_k$ are linearly related to the system state. The dynamic system is called **free** or **unforced** if there is no input, i.e. $u_k = 0$. A system with input is called a **forced** system.

**Example of state and measurement equation**

A cart is moving without friction along a straight road. We want to describe the behavior of the cart by an evolution equation, which describes what the state of the cart is compared to the previous state. Assume we know the external force $F$ acting on the cart. Because we know the acceleration as result of the external force (Newton’s law $F = ma$), we choose as state vector $x = \begin{pmatrix} s \\ v \end{pmatrix}_k$ with $s$ the travelled distance and $v$ the velocity. However, the choice of the state vector is not unique. Later in this section we will add a third component to the state vector, namely the acceleration. The input $u$ of the system is the external force, which by Newton’s law equals $ma$, with $m$ the mass and $a$ the acceleration. The state vector has dimension 2, the input vector dimension 1. We can describe the dynamics of the cart using the equations of uniformly accelerated motion:

$$
\begin{align*}
    s_t &= s_0 + v_0 t + \frac{1}{2} a t^2 \\
    v_t &= v_0 + at
\end{align*}
$$

(7.73)

where $t$ is the time interval. If we choose $t$ equal to one, we have:

$$
\begin{align*}
    s_{k+1} &= s_k + v_k + \frac{1}{2} a_k \\
    v_{k+1} &= v_k + a_k
\end{align*}
$$

(7.74)

If we put this in matrix form, and apply $u = ma$, we get the following recursive equation which describes the evolution of the state of the cart:

$$
\begin{pmatrix} s \\ v \end{pmatrix}_{k+1} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} s \\ v \end{pmatrix}_k + \frac{1}{m} \left( \begin{pmatrix} 1 \\ 0 \end{pmatrix} ight) u_k.
$$

(7.75)

This equation (which is called the state equation or evolution equation) may be solved, i.e. the values of $s_k$ and $v_k$ for $k > 0$ can be calculated, if we know the initial values $s_0$ and $v_0$ and the external force $u_k$ acting on the system for $k \geq 0$.

Now, we want to continue with the equation which describes the relation between the state of the cart and the measurement. The measurement is
obtained from a tachometer, which measures the travelled distance \((= y =\text{ measured distance})\). This makes the relation between the state \(x\) and the output \(y\) very simple:

\[
y_k = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} s \\ v \end{pmatrix}_k.
\]

(7.76)

This equation is called the measurement or observation equation.

If we now would be able to describe the noise of the system, we would be able to apply the Kalman filter and estimate the position and velocity of the cart, using a new measurement.

**Linear stochastic systems**

Usually when models are used to describe the relation between components, the components itself, or the relation is not exactly known. The initial state of a system, which is the state at \(k = 0\), is not always exactly known. In our example of the cart the initial state is quite certain if we start with

\[
\begin{pmatrix} s \\ v \end{pmatrix}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix},
\]

because we start with the center of the coordinate system chosen as the center of the cart. But if we choose as starting point a point in the world coordinate system, then this point is much more uncertain. This means, the value of the initial state vector cannot always be predicted. It takes values according to a certain probability, with other words the initial state vector is a stochastic variable.

Also, the relation between two successive states is not always precisely known. The behavior of the cart cannot be precisely modeled, because of small perturbations due to random elements, like slip. A solution for incorporating the fact that a model which describes the dynamics of a system never can be precise, is to add a component to represent the noise: a random vector. This gives us the following stochastic difference equation for the evolution of the state vector:

\[
x_{k+1} = A_k x_k + B_k u_k + \epsilon_k
\]

(7.77)

with \(x_k\) the state vector, \(A_k\) the matrix relating \(x_k\) to \(x_{k+1}\), \(u_k\) the input vector, \(B_k\) the matrix describing the influence of the input, and \(\epsilon_k\) a vector with random components.

Also, we should incorporate the fact that the relation between the measurements and the state cannot be precisely modelled. So, the measurements should be expressed as a linear combination of the state vector plus a random, additive measurement error, \(\delta_k\):

\[
y_k = C_k x_k + \delta_k.
\]

(7.78)

The combination of the two previous equations is called a **dynamic model** or a **state space model**. In this stochastic model the matrices are dependent on time. If they are independent, the system is called stationary.
7.6. EXPLICIT MODELS

Kalman filter

The Kalman filter gives an estimate of the state of a dynamic system from noisy measurements. It gives a recursive minimum variance estimate of the state of a system.

First of all, we will describe what is needed to be able to apply the Kalman Filter algorithm. Suppose we have the following linear discrete-time state space model:

\[
\begin{align*}
x_{k+1} &= A_kx_k + B_ku_k + \epsilon_k \\
y_k &= C_kx_k + \delta_k
\end{align*}
\]

with:

1. \(x_k\) an \(n\)-dimensional state vector to be estimated,
2. \(u_k\) an \(m\)-dimensional input vector,
3. \(y_k\) an \(p\)-dimensional measurement vector,
4. \(A_k\) an \(n\) by \(n\) dimensional known system matrix,
5. \(B_k\) an \(n\) by \(m\) dimensional known input matrix,
6. \(C_k\) an \(p\) by \(n\) dimensional known measurement matrix,
7. \(\epsilon_k\) assumed to be a white sequence with covariance matrix \(Q_k\) (A white sequence is defined as a sequence of zero-mean, uncorrelated random variables. So, all members have zero means and are mutually uncorrelated with all other members of the sequence.)
8. \(\delta_k\) measurement error, assumed to be a white sequence with covariance matrix \(R_k\)
9. \(\epsilon_k\) and \(\delta_k\) are assumed to have zero crosscorrelation.

Suppose further that the initial state vector \(x_0\) is a random vector with mean equal to \(E[x_0] = \bar{x}_0\) and with known covariance matrix (denoted by \(P_0\)) equal to

\[
P_0 = E[(x_0 - \bar{x}_0)(x_0 - \bar{x}_0)^T].
\]

If \(x_0\) is a one-dimensional vector, \(P_0\) is equal to \(E(x_0^2) - \bar{x}_0^2\), which is the variance of \(x_0\). So, \(P_0\) is a measure of the dispersion of \(x_0\) about \(\bar{x}_0\). Because \(P_0\) provides a statistical measure of the uncertainty in \(x_0\), it is called the \textbf{error covariance matrix}. The diagonal elements are the expected values of the square of the errors in the knowledge of the state components. The trace of \(P_0\) is the sum of the diagonal elements. The off-diagonal terms are indicators of the crosscorrelation between the components of \(x_0\).

And suppose further that \(x_0\) is independent of \(\epsilon_k\) and \(\delta_k\).

From mathematical viewpoint there is no distinction between a free system and a forced system. So, from now on we will assume that \(u_k = 0\).
First part of Kalman filter: propagation of state and errors Consider now the problem of predicting the state of a dynamic system in which the initial state vector $x_0$ is known (estimate $\hat{x}_0$). In other words, we want to predict the state at time $t_1$ using the old estimate $\hat{x}_0$. Because this prediction is based on the previous estimate before using the measurements at time $t_1$, we call the prediction an a priori estimate. We denote this by $\hat{x}_1^-$. So the "hat" denotes estimate and the "upper minus" denotes that this is our best estimate prior to assimilating the measurement at time $t_1$. The state equation could be used to form this prediction:

$$\hat{x}_1^- = A_0 \hat{x}_0. \tag{7.80}$$

We also want to propagate the error covariance matrix (denoted by $P_0^-$) associated with the a priori estimate, from time $t_0$ to $t_1$. One can easily compute that:

$$P_1^- = A_0 P_0 A_0^T + Q_0.$$

From this we see that the size of the random system disturbance ($Q_0$) has a direct bearing on the magnitude of the error covariance matrix at any point in time.

Second part of Kalman filter: update equations Given an a priori estimate $\hat{x}_1^- (= \text{prediction})$ of the system at time $t_1$, we seek an updated estimate $\hat{x}_1$ based on measurement $y_1$. The Kalman filter gives an estimate in linear (in $\hat{x}_1^-$ and $y_1$), recursive form, which blends the noisy measurement and the a priori estimate in the following way:

$$\hat{x}_1 = \hat{x}_1^- + K_1(y_1 - C_1 \hat{x}_1^-) \tag{7.81}$$

where $K_1$ is a weighting matrix, as yet unspecified. The problem is to find the blending factor $K_1$ that yields an updated estimate that is optimal in some sense. To come to this optimization criterion, we first have to define the error covariance matrix associated to the updated estimate, which is called the a posteriori error covariance matrix:

$$P_1 = E[(x_1 - \hat{x}_1)(x_1 - \hat{x}_1)^T]. \tag{7.82}$$

Returning to the optimization problem, we wish to find the particular $K_1$ which minimizes the individual terms along the diagonal of $P_1$, because these terms represent the estimation error variances for the elements of the state vector being estimated. The optimization is done by differentiating the trace of $P_1$ with respect to $K_1$. Then we find the optimal gain $K_1$ to be:

$$K_1 = P_1^- C_1^T(C_1 P_1^- C_1^T + R_1)^{-1}. \tag{7.83}$$
This matrix is called the optimal gain matrix. Using this gain matrix we find the relation between the a posteriori and a priori error covariance matrices:

\[ P_1 = (I - K_1 C_1) P_1^- \]  

(7.84)

This means that starting with an initial estimate and its associated error covariance matrix, we found an updated estimate and its associated error covariance matrix at time \( t_1 \). We can generalize this to time \( t_k \) in the following way.

So assume, we have an estimate of the state at some time \( t_k \), denoted by \( \hat{x}_k \). Then, we are able to find the prediction \( \hat{x}_{k+1}^- \) of the state at time \( t_{k+1} \), and the updated estimate \( \hat{x}_{k+1} \) based on use of measurement \( y_{k+1} \) in the following way. First, the prediction equations give us the prediction of the state with the corresponding error covariance matrix:

\[
\begin{align*}
\hat{x}_{k+1}^- &= A_k \hat{x}_k \\
P_{k+1}^- &= A_k P_k A_k^T + Q_k
\end{align*}
\]  

(7.85)

Then, the Kalman gain matrix \( K_{k+1} \) can be computed, which can be used to compute the updated estimate with the corresponding error covariance matrix:

\[
K_{k+1} = P_{k+1}^- C_{k+1}^T (C_{k+1} P_{k+1}^- C_{k+1}^T + R_{k+1})^{-1} \rightarrow \text{Kalman Gain matrix}
\]  

(7.86)

\[
\begin{align*}
\hat{x}_{k+1} &= \hat{x}_{k+1}^- + K_{k+1} (y_{k+1} - C_{k+1} \hat{x}_{k+1}^-) \\
P_{k+1} &= (I - K_{k+1} C_{k+1}) P_{k+1}^-
\end{align*}
\]  

(7.87)

It is clear that the Kalman filter algorithm is a recursive, linear estimator. The updated estimate is given by the prior estimate plus an appropriately weighted difference between the new measurement and its expected value, given by the prior estimate. The quantity \( y_{k+1} - C_{k+1} \hat{x}_{k+1}^- \) is called the measurement residual. The Kalman filter is a computer algorithm for processing discrete measurements into optimal outputs.

**Summary of Kalman filter equations**

| state equation | \( x_{k+1} = A_k x_k + \epsilon_k \)  
| measurement equation | \( y_k = C_k x_k + \delta_k \)  
| initial conditions | \( E[x_0] = \hat{x}_0, \ E[(x_0 - \hat{x}_0)(x_0 - \hat{x}_0)^T] = P_0 \)  
| other assumptions | \( E[\epsilon_i \epsilon_j^T] = 0 \) for all \( i, j \)  
| state prediction | \( \hat{x}_{k+1}^- = A_k \hat{x}_k \)  
| error covariance prediction | \( P_{k+1}^- = A_k P_k A_k^T + Q_k \)  
| Kalman gain | \( K_{k+1} = P_{k+1}^- C_{k+1}^T (C_{k+1} P_{k+1}^- C_{k+1}^T + R_{k+1})^{-1} \)  
| state update | \( \hat{x}_{k+1} = \hat{x}_{k+1}^- + K_{k+1} (y_{k+1} - C_{k+1} \hat{x}_{k+1}^-) \)  
| error covariance update | \( P_{k+1} = (I - K_{k+1} C_{k+1}) P_{k+1}^- \)  

Q_k \) is white (\( R_k \) is white)
Some remarks

1. It is often said that the Kalman filter generates its own error analysis. Clearly, this refers to the computation of the error covariance matrix $P_{k+1}$, which provides an indication of the accuracy of the estimate. The error covariance matrix is independent of the measurements, which means it can be computed in advance.

2. The larger the system disturbance (size of the $Q$-matrix), the more rapidly the error covariance increases. Increasing $Q$ would indicate either stronger noise driving the dynamics or increased uncertainty in the adequacy of the model itself. This will increase the rate of growth of the $P$ elements. As a result the filter gains will generally increase, thereby weighting the measurements more heavily.

3. Increasing $R$ would indicate that the measurements are subjected to a stronger corruptive noise, and so should be weighted less by the filter. This will decrease the gain values. The effect of the measurement noise on the error covariance matrix can be observed from the expression:

$$\frac{1}{P_k} = \frac{1}{P_k^*} + C_k^T R_k^{-1} C_k. \quad (7.88)$$

Large measurement noise ($R_k^{-1}$ is small) provides only a small increase in the inverse of the error covariance matrix. This is a small decrease in the error covariance matrix when the measurement is used. On the other hand small measurement error (large $R_k^{-1}$) cause the error covariance to decrease considerably whenever a measurement is used.

4. The optimality of the Kalman Gain matrix is contained in its structure and in the specification of the gain matrices. The meaning of the Kalman Gain matrix can be deduced from a simpler expression containing the a posteriori error covariance matrix:

$$K_k = P_k C_k^T R_k^{-1}. \quad (7.89)$$

Assume first that $C$ is the identity matrix. Then both $P$ and $R^{-1}$ are $n$ by $n$ matrices. If $R^{-1}$ is a diagonal matrix (no cross-correlation between noise terms), then $K$ results from multiplying each column of the error covariance matrix by the appropriate inverse of mean square measurement noise. Each element of the filter gain matrix is essentially the ratio between statistical measures of the uncertainty in the state estimate and the uncertainty in the measurement. Thus, the gain matrix is proportional to the uncertainty in the estimate and inversely proportional to the measurement noise. If measurement noise is large and state estimate errors are small, then $K$ is small. This means
the difference between the actual and predicted measurements will be used only for small corrections in the estimate. On the other hand, when measurement noise is small and the uncertainty in the estimate is large, $K$ will be large, and the difference will be used for strong corrections in the estimates.

5. If the noise is Gaussian (i.e. $\epsilon_k$ and $\delta_k$ are Gaussian), the filter gives the minimum variance estimate of the state, this means this is the best filter of any conceivable form. A nonlinear filter will not do better.

6. If the noise is not Gaussian, the filter gives the linear minimum variance estimate, this means this is the smallest among all linear estimates. So the Kalman filter is the best filter out of the class of linear filters.

Examples

To give an idea how the Kalman filter works, we give some examples which are related to previous examples. In the first example we try to estimate the position and velocity of a cart. The equations are based on the equations given in 7.79. We compute step by step the estimates and covariance matrices. In the next example we try to estimate the lengths of a two-link robot arm with the Kalman filter. This means that each time a new measurement is taken, we compute new estimates of the lengths. This time we only give the results, and don't work it out. In the last example we again estimate the position and velocity of a vehicle, but now we compare 4 different Kalman filters which use different kind of measurements. The results are given in two figures.

Example: Position and velocity

To get a feeling of how the Kalman filter works, we present a detailed computational example. We want to solve the problem of estimating the position and velocity of a cart which is moving without friction along a straight road. The deterministic evolution and measurement equation were given in 7.79. Let us assume that the evolution equation exactly describes how the system behaves, which means that the evolution equation doesn't contain a random component. Further, we assume that the relation between the state and the measurements is not exactly known. If we choose $m$ and $u_k$ equal to 1, we have the following dynamic model of the system:

\[
\begin{bmatrix}
    s_{k+1} \\
    v_{k+1}
\end{bmatrix} =
\begin{bmatrix}
    1 & 1 \\
    0 & 1
\end{bmatrix}
\begin{bmatrix}
    s_k \\
    v_k
\end{bmatrix} +
\begin{bmatrix}
    \frac{1}{2} \\
    1
\end{bmatrix}
\] (7.90)

\[
y_k =
\begin{bmatrix}
    1 & 0
\end{bmatrix}
\begin{bmatrix}
    s_k \\
    v_k
\end{bmatrix} + \delta_k.
\] (7.91)
If the initial position and velocity are exactly known, we can precisely compute the state of the system at any time \( k \). (So, we don’t need any measurements to compute the state of the system.) If the initial state is equal to:

\[
\begin{pmatrix}
    s \\
    v
\end{pmatrix}_0 = \begin{pmatrix}
    0 \\
    0
\end{pmatrix},
\]

then we can compute the state at time 1 as follows:

\[
\begin{pmatrix}
    s \\
    v
\end{pmatrix}_1 = \begin{pmatrix}
    1 & 1 \\
    0 & 1
\end{pmatrix} \begin{pmatrix}
    0 \\
    0
\end{pmatrix} + \begin{pmatrix}
    0.5 \\
    1
\end{pmatrix} = \begin{pmatrix}
    0.5 \\
    1
\end{pmatrix}.
\]

And:

\[
\begin{pmatrix}
    s \\
    v
\end{pmatrix}_2 = \begin{pmatrix}
    1 & 1 \\
    0 & 1
\end{pmatrix} \begin{pmatrix}
    0.5 \\
    1
\end{pmatrix} + \begin{pmatrix}
    0.5 \\
    1
\end{pmatrix} = \begin{pmatrix}
    2 \\
    2
\end{pmatrix}.
\]

We can compute the state at any time \( k \). In Table 7.3 the states are given for \( k = 0, \ldots, 10 \).

<table>
<thead>
<tr>
<th>( k )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_k )</td>
<td>0.0</td>
<td>0.5</td>
<td>2.0</td>
<td>4.5</td>
<td>8.0</td>
<td>12.5</td>
<td>18.0</td>
<td>24.5</td>
<td>32.0</td>
<td>40.5</td>
<td>50.0</td>
</tr>
<tr>
<td>( v_k )</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 7.3: Exact behavior during 10 steps.

Now we assume that the initial position and velocity of the cart are not exactly known. We want to estimate the position and the velocity of the cart using the measurements. At the end we shall compare the found estimates with the true values from Table 7.3.

Assume we have the measurements at time \( k = 1, \ldots, 10 \) given in Table 7.4.

Because the evolution equation is noise-free, we have \( Q = 0 \). We assume that the measurement noise is given by \( R = 1 \). Finally we assume that the initial state vector \( x_0 \) is a random vector with expectation and error covariance matrix respectively:

\[
Ex_0 = \hat{x}_0 = \begin{pmatrix}
    5 \\
    -1
\end{pmatrix} \quad \text{and} \quad P_0 = \begin{pmatrix}
    10 & 0 \\
    0 & 1
\end{pmatrix}.
\]

<table>
<thead>
<tr>
<th>( k )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y_k )</td>
<td>0.0</td>
<td>2.1</td>
<td>5.6</td>
<td>7.3</td>
<td>12.7</td>
<td>17.9</td>
<td>24.2</td>
<td>32.3</td>
<td>40.0</td>
<td>50.3</td>
</tr>
</tbody>
</table>

Table 7.4: Measurements during 10 steps.
We first compute the a priori estimate \( \hat{x}^{-1} \):

\[
\hat{x}^{-1} = A\hat{x}_0 + u_0 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 5 \\ -1 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 1 \end{pmatrix} = \begin{pmatrix} 4.5 \\ 0 \end{pmatrix}.
\]

Then we compute the a priori error covariance matrix \( P_1^{-1} \):

\[
P_1^{-1} = AP_0A^T + Q = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 11 & 1 \\ 1 & 1 \end{pmatrix}.
\]

In the third step we compute the gain matrix \( K_1 \):

\[
K_1 = P_1^{-1}C^T(CP_1^{-1}C^T + R)^{-1} = 
\begin{pmatrix} 11 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \left( \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + 1 \right)^{-1} = 
\begin{pmatrix} 11 \\ 1 \end{pmatrix}(11 + 1)^{-1} = \frac{1}{12} \begin{pmatrix} 11 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.9167 \\ 0.0833 \end{pmatrix}.
\]

Then we compute the updated estimate \( \hat{x}_1 \):

\[
\hat{x}_1 = \hat{x}_1^{-1} + K_1(y_1 - C\hat{x}_1^{-1}) = 
\begin{pmatrix} 4.5 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.9167 \\ 0.0833 \end{pmatrix} \left\{ 0.0 - \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 4.5 \\ 0 \end{pmatrix} \right\} = 
\begin{pmatrix} 4.5 \\ 0 \end{pmatrix} - 4.5 \begin{pmatrix} 0.9167 \\ 0.0833 \end{pmatrix} = \begin{pmatrix} 4.5 \\ 0 \end{pmatrix} - \begin{pmatrix} 4.125 \\ 0.375 \end{pmatrix} = \begin{pmatrix} 0.375 \\ -0.375 \end{pmatrix}.
\]

And the corresponding error covariance matrix \( P_1 \):

\[
P_1 = (I - K_1C)P_1^{-1} = 
\begin{pmatrix} 1.0000 \\ 0.0833 \end{pmatrix} \begin{pmatrix} 0.9167 \\ 0.0833 \end{pmatrix} \begin{pmatrix} 1.0000 \\ 0.0833 \end{pmatrix} \begin{pmatrix} 11 \\ 1 \end{pmatrix} = 
\begin{pmatrix} 0.9167 \\ 0.0833 \end{pmatrix} \begin{pmatrix} 11 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.9167 \\ 0.0833 \end{pmatrix} \begin{pmatrix} 11 \\ 1 \end{pmatrix} = 
\begin{pmatrix} 0.9167 & 0.0833 \\ 0.0833 & 0.9167 \end{pmatrix}.
\]

Now, we can continue to compute the a priori estimate \( \hat{x}_2^{-1} \), the corresponding covariance matrix \( P_2^{-1} \), the gain matrix \( K_2 \), the updated estimate \( \hat{x}_2 \) and \( P_2 \), etc. The results for 10 steps are given in table 7.5.

In figure 7.15 we compare the estimated values with the true values from table 7.3. On the horizontal axis the time is given. On the vertical axis the
Table 7.5: Measurement error, real minus estimated values and root mean square error during 10 steps.

measurement errors are plotted, the errors in the position estimates $s_k - \hat{s}_k$ and the errors in the velocity estimates $v_k - \hat{v}_k$. The standard deviation of the errors (also called root mean square (rms) error) is given in table 7.5, which is the root of the error covariance of the estimates. As soon as the first observation is processed, the root mean square error drops from 3.162 to 0.957. The first observation helps us to increase the precision of the initial estimate, which was rather small. On the other hand, the rms error of the velocity does not decrease very much after the first observation is processed. This is because two position measurements are required to determine the velocity. Because the velocity affects the position, the velocity must first be estimated rather accurately before accurate position estimates can be obtained.

Example: Inverse kinematics

Suppose we want to estimate the lengths $\ell_1$ and $\ell_2$ of a two-link robot arm. Instead of computing the minimum least-squares estimate, as done in paragraph 7.6.1, we want to compute the minimum variance estimate of the lengths with the Kalman filter. This means we want to estimate two random constants $\ell_1$ and $\ell_2$ based on a sequence of independent noisy measurements of the positions $(x_k, y_k)$ of the gripper which corresponds to a sequence of joint values $(\theta_{1k}, \theta_{2k})$ for $k = 1, 2, \ldots, N$. Each time we have a new measurement, we want to update the estimates of the lengths.

Because we have to distinguish the $x$-coordinate of the positions from the state vector which is usually notated as $x$, we use the bold $\mathbf{x}$ for the
state vector. The state vector, consisting of the values to be estimated, is:

\[ \mathbf{x}_k = \begin{pmatrix} \ell_1 \\
\ell_2 \end{pmatrix}_k. \]

The constants \( \ell_1 \) and \( \ell_2 \) satisfy the difference equation:

\[ \mathbf{x}_{k+1} = \mathbf{x}_k. \tag{7.92} \]

Because the lengths are constants, \( Q_k = 0 \). After each move of the robot, a new measurement is taken and based on the measurement a new estimate has to be computed. The relation between the position measurements \((x_k, y_k)\) and the lengths \( \ell_1 \) and \( \ell_2 \) is given in the following measurement equation:

\[ \begin{pmatrix} x \\
y \end{pmatrix}_k = \begin{pmatrix} \cos \theta_1 & \cos(\theta_1 + \theta_2) \\
\sin \theta_1 & \sin(\theta_1 + \theta_2) \end{pmatrix}_k \begin{pmatrix} \ell_1 \\
\ell_2 \end{pmatrix}_k + \begin{pmatrix} \delta_1 \\
\delta_2 \end{pmatrix}_k. \tag{7.93} \]

Further, we assume that the error variance of both components of the position measurements is equal to 1. So we have:

\[ A = I, Q = 0 \]

and

\[ C_k = \begin{pmatrix} \cos \theta_{1k} & \cos(\theta_{1k} + \theta_{2k}) \\
\sin \theta_{1k} & \sin(\theta_{1k} + \theta_{2k}) \end{pmatrix}, R = \begin{pmatrix} 1 & 0 \\
0 & 1 \end{pmatrix}. \]
Finally, we assume that the initial state vector $x_0$ is a random vector with expectation and error covariance matrix respectively:

$$E x_0 = \hat{x}_0 = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix} \quad \text{and} \quad P_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$  

In this example the measurement matrix $C_k$ is dependent on the joint angles $\theta_{1k}$ and $\theta_{2k}$. This means that the error covariance matrix $P_k$ and the Kalman gain matrix $K_k$ cannot be computed in advance. Each time a new measurement is taken, these matrices have to be recalculated. In table 7.6 the joint angles and the corresponding the exact $(x,y)$ coordinates of 20 positions are given. These are based on the "exact" arm lengths equal to 1. Further, the corresponding measured positions and the estimates of the two arm lengths are given.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$\theta_1$</th>
<th>$\theta_2$</th>
<th>$x_{real}$</th>
<th>$y_{real}$</th>
<th>$x_{meas}$</th>
<th>$y_{meas}$</th>
<th>$\hat{\ell}_1$</th>
<th>$\hat{\ell}_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>2.000</td>
<td>0.000</td>
<td>2.106</td>
<td>0.121</td>
<td>1.202</td>
<td>1.202</td>
</tr>
<tr>
<td>2</td>
<td>0.000</td>
<td>0.524</td>
<td>1.866</td>
<td>0.500</td>
<td>1.947</td>
<td>0.532</td>
<td>1.139</td>
<td>1.144</td>
</tr>
<tr>
<td>3</td>
<td>0.000</td>
<td>1.047</td>
<td>1.500</td>
<td>0.866</td>
<td>1.592</td>
<td>1.024</td>
<td>1.101</td>
<td>1.159</td>
</tr>
<tr>
<td>4</td>
<td>0.000</td>
<td>1.571</td>
<td>1.000</td>
<td>1.000</td>
<td>1.074</td>
<td>1.045</td>
<td>1.108</td>
<td>1.133</td>
</tr>
<tr>
<td>5</td>
<td>0.000</td>
<td>2.094</td>
<td>0.500</td>
<td>0.866</td>
<td>0.569</td>
<td>1.031</td>
<td>1.111</td>
<td>1.137</td>
</tr>
<tr>
<td>6</td>
<td>0.000</td>
<td>2.618</td>
<td>0.134</td>
<td>0.500</td>
<td>0.206</td>
<td>0.593</td>
<td>1.124</td>
<td>1.127</td>
</tr>
<tr>
<td>7</td>
<td>0.000</td>
<td>3.142</td>
<td>0.000</td>
<td>0.000</td>
<td>0.017</td>
<td>0.078</td>
<td>1.127</td>
<td>1.124</td>
</tr>
<tr>
<td>8</td>
<td>0.524</td>
<td>0.000</td>
<td>1.732</td>
<td>1.000</td>
<td>1.759</td>
<td>1.122</td>
<td>1.110</td>
<td>1.108</td>
</tr>
<tr>
<td>9</td>
<td>1.047</td>
<td>0.000</td>
<td>1.000</td>
<td>1.732</td>
<td>1.105</td>
<td>1.787</td>
<td>1.100</td>
<td>1.098</td>
</tr>
<tr>
<td>10</td>
<td>1.571</td>
<td>0.000</td>
<td>0.000</td>
<td>2.000</td>
<td>0.196</td>
<td>2.065</td>
<td>1.091</td>
<td>1.088</td>
</tr>
<tr>
<td>11</td>
<td>2.094</td>
<td>0.000</td>
<td>-1.000</td>
<td>1.732</td>
<td>-0.973</td>
<td>1.848</td>
<td>1.085</td>
<td>1.083</td>
</tr>
<tr>
<td>12</td>
<td>2.618</td>
<td>0.000</td>
<td>-1.732</td>
<td>1.000</td>
<td>-1.688</td>
<td>1.136</td>
<td>1.077</td>
<td>1.075</td>
</tr>
<tr>
<td>13</td>
<td>3.142</td>
<td>0.000</td>
<td>-2.000</td>
<td>0.000</td>
<td>-1.830</td>
<td>0.021</td>
<td>1.061</td>
<td>1.059</td>
</tr>
<tr>
<td>14</td>
<td>0.524</td>
<td>0.524</td>
<td>1.366</td>
<td>1.366</td>
<td>1.488</td>
<td>1.458</td>
<td>1.063</td>
<td>1.060</td>
</tr>
<tr>
<td>15</td>
<td>1.047</td>
<td>1.047</td>
<td>0.000</td>
<td>1.732</td>
<td>0.012</td>
<td>1.816</td>
<td>1.063</td>
<td>1.058</td>
</tr>
<tr>
<td>16</td>
<td>1.571</td>
<td>1.571</td>
<td>-1.000</td>
<td>1.000</td>
<td>-0.868</td>
<td>1.014</td>
<td>1.065</td>
<td>1.046</td>
</tr>
<tr>
<td>17</td>
<td>2.094</td>
<td>2.094</td>
<td>-1.000</td>
<td>0.000</td>
<td>-0.922</td>
<td>0.016</td>
<td>1.063</td>
<td>1.043</td>
</tr>
<tr>
<td>18</td>
<td>2.618</td>
<td>2.618</td>
<td>-0.366</td>
<td>-0.366</td>
<td>-0.298</td>
<td>-0.361</td>
<td>1.057</td>
<td>1.047</td>
</tr>
<tr>
<td>19</td>
<td>3.142</td>
<td>3.142</td>
<td>0.000</td>
<td>0.000</td>
<td>0.055</td>
<td>0.176</td>
<td>1.053</td>
<td>1.052</td>
</tr>
<tr>
<td>20</td>
<td>0.785</td>
<td>1.047</td>
<td>0.448</td>
<td>1.673</td>
<td>0.530</td>
<td>1.869</td>
<td>1.058</td>
<td>1.055</td>
</tr>
</tbody>
</table>

Table 7.6: Estimates of lengths of robotarm during 20 steps.

In figure 7.16 the estimation errors are shown. It is clear that the initial estimates of the lengths are to high (1.5 instead of 1). The Kalman filter immediately reacts on the first measured position by decreasing the first estimates to 1.2. Then, subsequent measurements gradually improve the estimates until they reach the value of about 1.05. Because the uncertainty in
the state propagation is small ($Q = 0$), the Kalman gain matrix is becoming smaller and smaller. The filter is weighting the measurement residuals less and less, which at the end doesn’t lead to better results.

We should compare the results of the Kalman filter with the parameter estimation example in section 7.6.1 in which the lengths $\ell_1$ and $\ell_2$ are estimated based on $N$ measured positions $(x_i, y_i)$ corresponding to joint values $(\theta_{1i}, \theta_{2i})$. In this example we worked out the formulas for computing the least square solution that minimized the quadratic error:

$$E = \sum_{i=1}^{N} d_i^2 = \sum_{i=1}^{N} \left( \{x_i - x(\theta_{1i}, \theta_{2i})\}^2 + \{y_i - y(\theta_{1i}, \theta_{2i})\}^2 \right).$$  \hspace{1cm} (7.94)

The solution was computed as a function of the values $S_1$, $S_2$ and $P$:

$$\ell_1 = \frac{NS_1 - PS_2}{N^2 - P^2},$$

$$\ell_2 = \frac{NS_2 - PS_1}{N^2 - P^2}. \hspace{1cm} (7.95)$$

The measured positions were taken from table 7.6. These and the corresponding joint values are used to compute the values $S_1$, $S_2$ and $P$. These result in: $S_1 = 26.513$, $S_2 = 26.469$ and $P = 5.5$. From that it follows that: $\ell_1 = 1.04039$ and $\ell_2 = 1.03732$. These values are of course different then in the Kalman solution, because different values are minimized. In this example the cost function $E = \sum_{i=1}^{N} d_i^2$ is minimized, and in the Kalman filter solution the error covariance. The result of the Kalman filter is worse,
because we assumed quite a large measurement noise (with variance equal to 1). It turns out that if we assume the error variance of the measurements is equal to 0.01, the estimates of the lengths become 1.04044 and 1.03739, which is comparable to the least-squares solution, but again not the same.

**Example: Position and velocity with different kind of measurements**

If we combine the measurements of sensors of different nature, which can be done with the Kalman filter, a system is getting more robust. In this way different sorts of errors can be recognized and corrected for. In the following example a combination of shaft encoders with ultrasonic sensors will be used to estimate the position of a one-motor vehicle. The ultrasonic sensors have more noise than the shaft encoders, and the shaft encoders have a larger systematic error than the ultrasonic sensors. The combination of these two sensors of different nature makes it possible to minimize the systematic error of the shaft encoder. The combination of these two sensors will give us a better estimate of the position than could be obtained from either the ultrasonic sensor or the shaft encoder alone.

We want to estimate the position of a vehicle which is moving along a straight line. We compare 4 different Kalman filters using different kind of measurements. The sensors we want to use are:

1. distance sensor (like an ultrasonic sensor) which has large random errors,

2. shaft encoder (incremental encoder) which has small random errors, but has a systematic error.

We combine 4 Kalman Filters which use the following (combination of) sensors:

1. distance sensor
2. shaft encoder
3. shaft encoder and distance sensor combined
4. same combination but in an Extended Kalman Filter (non-linear Kalman Filter in which the equations are linearized).

**Example 1: distance sensor** We choose as state vector

\[ x_k = \begin{pmatrix} s \\ v \\ a \end{pmatrix}_k \]
in which the acceleration is included because we don’t know the force acting on the vehicle. In the first three examples this state vector will be used. In the input vector we describe how the acceleration changes, which we compute from the difference in voltage and speed. We don’t include the input vector in the equations, because the vector only forms an offset which has to be added to the Kalman Filter equations. The evolution equation is obtained from the equations for uniformly accelerated motion:

\[
\begin{align*}
    s_{k+1} &= s_k + v_k + \frac{1}{2}a_k \\
    v_{k+1} &= v_k + a_k
\end{align*}
\]

which gives if we write this in matrix form:

\[
\begin{pmatrix}
    s \\
    v \\
    a
\end{pmatrix}_{k+1} =
\begin{pmatrix}
    1 & 1 & \frac{1}{2} \\
    0 & 1 & 1 \\
    0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
    s \\
    v \\
    a
\end{pmatrix}_k + \epsilon_k. \quad (7.96)
\]

This evolution equation will be used for the first three (linear) Kalman filters. In the first measurement equation the outcome of the distance sensor is compared to the travelled distance:

\[
(s_{\text{dist}})_k = \begin{pmatrix}
    1 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
    s \\
    v \\
    a
\end{pmatrix}_k + (\delta_1)_k.
\]

This means that for the first Kalman filter we have defined the state equation and the measurement equation, i.e. the state vector and the matrices \( A \) and \( C \). If we now further define the system covariance matrix \( Q \), the measurement covariance matrix \( R \) and the initial covariance matrix \( P_0 \), we can apply the Kalman filter equations in the summary. So, starting with an initial estimate of the state vector \( \hat{x}_0 \), we can find a prediction of the state \( \hat{x}_1^- \) and the corresponding error covariance matrix \( P_1^- \). Then, the Kalman gain matrix \( K_1 \) can be computed. Then, the new measurement \( y_1 \) can be used to update the estimate and obtain \( \hat{x}_1 \). Also, the corresponding error covariance matrix \( P_1 \) can be computed. At the next iteration we use the updated estimates \( \hat{x}_1 \) and \( P_1 \) to compute \( \hat{x}_2 \) and \( P_2 \), etc.

**Example 2: shaft encoder** In example 2 the same state vector and evolution equation will be used. In the measurement equation we want to compare the outcome of the shaft encoders. But because the shaft encoder has a systematic error, it is better to use the difference of the successive shaft encoder measurements. Then, the systematic error disappears, and the measurement vector is the measured velocity \( v_{\text{shaft}} \). Because the difference of two successive measurements gives us the velocity at the middle of the
time interval, we have to correct the velocity in the state vector with half of the acceleration. This gives us the following measurement equation:

\[
(v_{shaft})_k = \begin{pmatrix} 0 & 1 & -\frac{1}{2} \end{pmatrix} \begin{pmatrix} s \\ v \\ a \end{pmatrix}_k + (\delta_2)_k.
\]

For the second Kalman filter the same procedure is followed as described for the first Kalman filter to obtain the estimates of the state of the system.

**Example 3: distance sensor plus shaft encoder** In example 3 we want to combine the distance measurement of example 1 with the shaft encoder measurements in example 2. This gives us a two-dimensional measurement vector and a 2 by 3 measurement matrix:

\[
\begin{pmatrix} s_{dist} \\ v_{shaft} \end{pmatrix}_k = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -\frac{1}{2} \end{pmatrix} \begin{pmatrix} s \\ v \\ a \end{pmatrix}_k + \begin{pmatrix} \delta_1 \\ \delta_2 \end{pmatrix}_k.
\]

**Example 4: distance sensor plus shaft encoder in EKF** The problem with the shaft encoders is that it has a random error and a systematic error, which cannot be distinguished in the previous Kalman Filters. If it is possible it has an advantage to split the random and the systematic error and to model the errors explicitly. The source of the systematic error in shaft encoders is the scale factor which translates the shaft encoder counts in distance (mm's) or vice versa. Why not put this scale factor in the state vector and estimate in this way this factor. Because we have to relate the scale factor to components in the state vector, we add another component. We choose the velocity based on the difference of the shaft encoders in shaft encoders counts, denoted by \(v_s\). This means we have as state vector:

\[
\begin{pmatrix} s \\ v \\ a \\ v_s \\ scale \end{pmatrix}_k
\]

with \(scale\) the scale factor to convert distance (mm's) to shaft encoder counts and \(v_s\) the velocity (in shaft encoder counts) as result of the change in the actual position:

\[
(v_s)_{k+1} = scale_k * (s_{k+1} - s_k)
\]

The position can be related to the velocity and the acceleration of the state vector by:

\[
s_{k+1} - s_k = v_k + \frac{1}{2}a_k.
\]
We assume the scale factor to be constant. This gives us the following evolution equation:

\[
\begin{pmatrix}
    s \\
    v \\
    a \\
    v_s \\
    \text{scale}
\end{pmatrix}_{k+1} = \begin{pmatrix}
    1 & 1 & \frac{1}{2} & 0 & 0 \\
    0 & 1 & 1 & 0 & 0 \\
    0 & 0 & 1 & 0 & 0 \\
    \vdots & \vdots & \vdots & \vdots & \vdots \\
    0 & 0 & 0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
    s \\
    v \\
    a \\
    v_s \\
    \text{scale}
\end{pmatrix}_k + \alpha_k, \tag{7.97}
\]

where on the fourth row we have:

\[(v_s)_{k+1} = \text{scale} \times (v_k + \frac{1}{2}\alpha_k).\]

Of course, we should not give the evolution equation in matrix form because it is not linear in the components of the state vector. But we want to repeat the rows of the matrix which are the same as in example 3. In the measurement equation we relate the same measurements to the 5-dimensional state vector, in which case it seems to be more natural to relate the measured shaft encoder velocity to the actual shaft encoder velocity to be estimated:

\[
\begin{pmatrix}
    v_{shaft} \\
    v_{dist}
\end{pmatrix}_k = \begin{pmatrix}
    1 & 0 & 0 & 0 & 0 \\
    0 & 0 & 0 & 1 & 0
\end{pmatrix} \begin{pmatrix}
    s \\
    v \\
    a \\
    v_s \\
    \text{scale}
\end{pmatrix}_k + (\delta)_k.
\]

The way we implemented this is that we developed a simulation of the behavior of the cart. The setpoints of the desired behavior form the input to the simulation, the measurements are the output which are input to the Kalman filter. See figure 7.17. To generate the "real" values (position, velocity etc.) a dynamic model of the behavior of the cart is used with an additive noise component. To simulate the random character of the measurements random function generators were used. The simulation of the cart is treated as a black box, so the Kalman filter only sees the measurements and not the model and noise used in the simulation of the cart. The output of the Kalman Filter are the estimates of the state variables. These were compared to the actual values to obtain an idea of the capabilities of the Kalman filter.

In figures 7.18 and 7.19 the results of the 4 Kalman Filters are compared.

On the horizontal axis the time is given, and the vertical axes the difference between the actual value and the estimated value (which in the ideal situation should be zero). From the error in the position of the first Kalman filter (only distance sensor) it is clear that the Kalman filter is able to react on the random noise present in the measurement of the distance sensor, but is not able to give a satisfactory solution. From the second example it is
Figure 7.17: Diagram of implementation of Kalman filter.

Figure 7.18: Comparison of position estimate of 4 Kalman filters.
clear, that the systematic error of the shaft encoder adds up in the position estimate and the Kalman filter is not able to recognize the failure of the shaft encoder. After about 120 time steps the velocity is almost zero and the systematic error is not adding up any more. From the third example we can see that the Kalman filter first is distracted by the systematic error. But then it is able to handle the situation, but it cannot forget its starting failure. As expected, the Extended Kalman filter has the best results. The main reason is that we modeled the systematic error as a component of the state vector. This enabled the filter to recognize this error and separate it from the random errors.

In figure 7.19 it can be seen that the estimate of the velocity is acceptable for the shaft encoder. This because in the estimate of the velocity the 'measured shaft encoder velocity' is used in which the systematic error is subtracted. The error in the velocity estimate corresponding to the distance sensor is increased because the random disturbances can add up. The behavior of the third example is the same as the second, which means that the combination of the shaft encoder and the distance sensor does not help in estimating the velocity. The extended Kalman filter again is the best.
7.7 Implicit models

As an example of an implicit model, the previous section presented a feedforward network to represent the inverse kinematics of a robot arm. Such a network is able to represent the function relating the joint angles to the world coordinates.

In the course on neural networks [12] various other network types have been described for function approximation, such as counterpropagation networks, radial basis functions etc. They all are of the form

\[ y_h = \sum_{i=1}^{K_{\text{hidden}}} w_{hi} \phi \left( \sum_{j=1}^{K_{\text{in}}} w_{ij} x_j + w_{i0} \right) + w_{h0}, \quad (7.98) \]

in which \( \phi(\cdot) \) is a gaussian shaped function for radial basis function networks or a delta function for table-lookup networks or a non-overlapping region in input space for counterpropagation networks.

Apart from these networks which are suited for function approximation, there is a class of networks which is suited for non-linear projections, such as the Kohonen networks. The Kohonen algorithm learns from examples a mapping from the input space \( \mathcal{X} \) to a lattice \( \mathcal{A} \) of \( N \) formal neurons \( r_1, \ldots, r_N \). The map \( \phi_w : \mathcal{X} \rightarrow \mathcal{A} \) assigns to each input vector \( x \in \mathcal{X} \) an element \( \phi_w(x) \in \mathcal{A} \), and is defined by the “weights” \( w = (w_1, w_2, \ldots, w_N) \), \( w_i \in \mathcal{X} \). The function \( \phi_w(x) \) is specified by the condition

\[ |w_{\phi_w(x)} - x| = \min_{r \in \mathcal{A}} |w_r - x|, \quad (7.99) \]

i.e., a vector \( x \) is mapped onto that neuron \( r \) for which \( |w_r - x| \) becomes minimal.

In order to determine the approximation which represents the systems behavior (or its inverse) as accurate as possible, we have to estimate the parameters of the model on the basis of measurements from the real world. The previous section presented two examples in which the system is linear in its parameters and the error is quadratic in its parameters. A least square error solution can be found analytically. Most neural networks are nonlinear functions of the parameters (‘weights’) and the best solution can not be found analytically. Usually gradient descent methods are used.

7.7.1 Example 1: Train a neural network to represent the inverse kinematics.

In this example we use a feed-forward network with \( K \) hidden units as given in equation 7.40. For the learning procedure a set of \( N \) learning samples \( (x_p, y_p, \theta_{1,p}, \theta_{2,p}) \) is available. The error criterion is the sum of the square
error at the output of the network:

\[ E = \sum_{p=1}^{N} E_p \]
\[ = \sum_{p=1}^{N} (\theta_{1,p} - \hat{\theta}_{1,p})^2 + (\theta_{2,p} - \hat{\theta}_{2,p})^2, \]

in which \( E_p \) represents the square error on pattern \( p \) and \( \hat{\theta}_{1,p} \) and \( \hat{\theta}_{2,p} \) are the network outputs given by equation 7.40. Note that in contrast to the example given in the previous section, where the parameters were estimated minimizing the error of the forward model, here the neural approach minimizes the output error of the inverse model which can be used as controller.

\( E \) has to be minimized with respect to the parameters \( w_{ij} \). In a gradient descent algorithm the idea is to change the parameters proportional to the negative of the derivative of the error as measured on the current pattern with respect to each weight:

\[ \Delta_p w_{ij} = -\gamma \frac{\partial E_p}{\partial w_{ij}} \]

where \( \gamma \) is a constant of proportionality (learning rate). The procedure how the weights are updated and how the error at the output is ‘back-propagated’ to the hidden layer is described more extensively in the course on neural networks [12].

After a large enough number of iterations the error does not decrease anymore and a solution is found which gives the minimal error over the learning samples. Because of the nonlinear relation between \( E \) and the weights \( w_{ij} \), there is a danger of ending in a local minimum. However, because the number of ‘free’ parameters is much higher than the number of intrinsic parameters, a good solution is usually found.

### 7.7.2 Model validation

In contrast to the explicit model based approach, the \( \chi^2 \) method can not be used to test the goodness of the model. The point is that we can always find a neural network which fits the learning samples. The important question is how many hidden units are needed in the network.

If a too small number of units is taken, the network is not able to represent the function accurate enough. If too many units are taken, there will be overfitting; the individual learning samples will be represented accurately, but generalisation will be poor. It is important to test the performance with an independent testset. If the error on the testset starts increasing when more hidden units are added (“peaking effect”) the network is too large.
7.7.3 Example 2: Train a Kohonen network to represent the free space of an environment

In the previous section it was discussed that space can be represented as a graph structure. In order to digitize the continuous space and to preserve topological relations, a Kohonen network can be used. If the input samples are drawn from the free space of the environment, for example by random exploration of a robot, the Kohonen network will form a discrete representation of that free space while the neighbouring relations are preserved in the topology of the network. In the following example a Kohonen network is used as discrete representation of the 2D free space of a mobile robot. The input of the network is formed by the position of the mobile robot.

The vehicle is positioned randomly in the free space. During learning the weights of all neurons \( r \) in the network are adapted according to the Kohonen rule

\[
\Delta w_r = \eta \Lambda(r, k)(x - w_r),
\]  

(7.103)

where the learning parameter \( \eta \) is multiplied with \( \Lambda(r, k) \), a decreasing function of distance of neuron \( r \) to the winner \( k \) [11]. Learning also decreases with time to obtain a stable network.

After a sufficiently large number of examples, the Kohonen map gives a discretized version of the free space. It has to be noted that a good discretization is only obtained in case of convex spaces. If the space is not convex, neurons may be placed in “forbidden” area’s (see figure 7.20). Different procedures such as the “neural gas” have been presented which are able to overcome this problem.
Figure 7.20: Kohonen map of a non-convex room after training. Note that the Kohonen map has neurons in “occupied” space.
Bibliography


