Chapter 3

Learning sensor models

3.1 Introduction

In section **??** we introduced the architecture of our Sensor Data Fusion (SDF) system. One of the main characteristics of that architecture is that each sensor first *converts* its measurements to a common internal representation before the actual fusion is performed. This is to facilitate fusion between sensors from different modalities. Such a system can also easily be extended with new sensors as long as these sensors convert their measurements to the same common internal representation. Also we argued that such an organization of an SDF system leads to a fault-tolerant system that gracefully degrades upon sensor failure.

While these are all very nice properties for an SDF system, until now we did not pay attention to the problem of converting the sensor measurements to the common internal representation. Indeed it is generally acknowledged (see, e.g., [21]) that this is a key issue in such SDF systems. Not only should the conversion include a mere "translation" from measurement to common internal representation, but as we argued in section **??** also the *certainty values* for the translated measurement should be given. These certainty values express the error characteristics of the sensor. They are vital to the subsequent fusion process; without certainty values the fusion process could not exploit the specific strengths (e.g. accurate measurement of distance) and weaknesses (e.g. inaccurate measurement of angle) of the sensors. Thus it is of great importance that the conversion of sensor measurements also includes a model of the sensor's error characteristics. In this chapter we will discuss how such models can be used to convert sensor measurements.

With the definitions given in the previous chapter we can already state the problem of converting sensor measurements to the common internal representation for our autonomous mobile robot more formally as:

> Given a sensor measurement $\boldsymbol{s}$ define **conversion functions** $\mathcal{G}_j$ such that
>
> $$g_j \triangleq P(\text{cell } c_j \text{ is occupied} \mid \boldsymbol{s}) = \mathcal{G}_j(\boldsymbol{s}), \qquad (3.1.1)$$
>
> for all cells c_j in the occupancy grid.

These conditional probabilities can then be used directly in our higher level fusion method, the PDOP (see (**??**), chapter **??**, section **??**).

In the definition of the PDOP it can be seen that the higher level fusion method also needs the joint conditional probabilities $P(\text{cell } c_j \text{ is occupied} | \boldsymbol{s}_1, \boldsymbol{s}_2)$. For ease of discussion, we will ignore these probabilities for the time being and focus on the conversion of single sensor measurements. Once we have

finished discussing these we will return to the point of joint conditional probabilities.

We commence our discussion of conversion functions with a review of conventional methods. We discuss the conversion functions which are most often used with an occupancy grid as common internal representation. We then review some variations to these basic conversion functions. Our main objection to all these approaches is that they only take into account the sensor specific characteristics (such as, e.g., the opening angle of the sensor), while the conversion functions are equally influenced by the environment. E.g., time-of-flight range sensors are also influenced by air temperature [16] and the specularity of the obstacles in the robot's environment [9]. One can imagine that also different conversion functions are preferable for, e.g., cluttered environments and hallways.

Obviously such complex environmental influences are hard to model explicitly with analytical expressions. Therefore we apply a neural network to *learn* the conversion functions from sensor measurements to probabilities in the occupancy grid. We define the neural network topology, the learning rule, and we introduce the *sense and drive* algorithm which provides the learning samples for the network. Experiments show that indeed these learned conversion functions are adaptive not only to changes in the sensor specific parameters but also to changes in the environment. We conclude this chapter with a discussion which puts our approach to learning the conversion functions in the context of our sensor data fusion system. While the conversion methods reviewed in this chapter are all predominantly suited for conversion to probabilistic grids, a review of conversion functions for Dempster-Shafer grids is also included in the discussion.

In this chapter we will restrict ourselves to sensors for which the measurements $\boldsymbol{s}$ are range measurements (e.g. acoustic sensors or infra-red range sensors). Most of the theory presented, however, is easily extendible to other types of sensors.

3.2 Conversion functions reviewed

In the discussion of conversion functions $\mathcal{G}_j$ we will derive a general formula for all cells c_j. We will use the following shorthands:

$\boldsymbol{s}$	distance measurement from sensor
r	true distance
$P(\text{OCC}(c_j))$	$P(\text{cell } c_j \text{ is occupied})$
$P(\text{EMP}(c_j))$	$P(\text{cell } c_j \text{ is empty})$

3.2.1 Basic probabilistic conversion functions

The most basic conversion functions $\mathcal{G}_j$, which are frequently used to convert sensor measurements to occupancy grids, are introduced by Elfes in, e.g., [5, 6]. The conditional probabilities in the grid are obtained using Bayes' rule:

$$P(\text{OCC}(c_j) \mid \boldsymbol{s}) = \frac{p(\boldsymbol{s} \mid \text{OCC}(c_j))P(\text{OCC}(c_j))}{P(\boldsymbol{s})}. \tag{3.2.1}$$

In this equation $p(\boldsymbol{s} \mid \text{OCC}(c_j))$ is called the **sensor model**. Hence the conditional probability $P(\text{OCC}(c_j) \mid \boldsymbol{s})$ is called the **inverse sensor model** (we already encountered the inverse sensor model in a general discussion of Bayesian fusion methods in section **??**). The probabilities $P(\text{OCC}(c_j))$ and $P(\boldsymbol{s})$ are the prior probabilities.

We will first explain how the terms in this equation are obtained by Elfes and we will then discuss our opinion about this particular conversion method.

The sensor model. Many approaches in the sensor data fusion literature are concerned with the definition of an accurate sensor model. In the standard approach presented in [5, 6] the sensor model $p(\boldsymbol{s} \mid \text{OCC}(c_j))$ is derived from its more general 'continuous' form $p(\boldsymbol{s} \mid r)$. This is a probability density function which gives the probability of a measurment $\boldsymbol{s}$ given the true value r. Typically, and also in [5, 6], Gaussian functions are used to define such probability densities. It can be easily understood intuitively how this Gaussian probability density function can be used to express the uncertainty and the measurement error of the sensor. An accurate and certain sensor would have a narrow Gaussian density; the ideal range sensor would be characterized by a continuous sensor model $\delta(\boldsymbol{s} - r)$, where $\delta()$ is the δ-Dirac function. Inaccurate sensors would have wider Gaussians.

The sensor model is related to its continuous form as follows:

$$p(\boldsymbol{s} \mid r) = p(\boldsymbol{s} \mid \text{OCC}(c_j) \text{ and } \text{EMP}(c_k) \text{ for cells } c_k \text{ 'closer' than } r). \tag{3.2.2}$$

This means that at distance r, in cell c_j an obstacle is present while all cells c_k between this cell and the sensor are empty, for else an obstacle in cell c_k would have caused a measurement $\boldsymbol{s}' < \boldsymbol{s}$.

This equation can now be used to estimate the sensor model. The estimation is performed by summing over all possible occupancy grid configurations using Kolmogoroff's theorem (see [6] for more details):

$$p(\boldsymbol{s} \,|\, \mathrm{OCC}(c_j)) = \sum_{\{G^{\mathrm{OCC}(c_j)}\}} \left(p(\boldsymbol{s} \,|\, G^{\mathrm{OCC}(c_j)}) \times P(G^{\mathrm{OCC}(c_j)}) \right). \quad (3.2.3)$$

In this equation $G^{\mathrm{OCC}(c_j)}$ represents an occupancy grid configuration for which cell c_j is occupied.

The probabilities $p(\boldsymbol{s} \mid G^{\mathrm{OCC}(c_j)})$ can be obtained with (3.2.2). The distribution of possible configurations $G^{\mathrm{OCC}(c_j)}$ could reflect the particular environment the robot is working in. In fact, this is precisely what we will do later on in this chapter. In the standard approach of Elfes, however, the distribution is derived from the individual prior cell occupancy probabilities $P(\mathrm{OCC}(c_j))$. This is a legitimate approach since the individual probabilities are assumed to be independent. The choice of these priors is discussed in more detail in the next paragraph.

The prior cell occupancy probabilities. The prior individual cell occupancy probabilities $P(\mathrm{OCC}(c_j))$ represent the a-priori probability of a cell being occupied; i.e., prior to the sensor measurement. In section **??** we have already discussed the importance of choosing accurate priors for the initial occupancy grid representation of the environment. The priors are also used in the conversion of sensor measurements with Bayes' rule (3.2.1) and, as we saw in the previous paragraph, in the calculation of the sensor model (3.2.3).

Ideally, the prior should reflect the average 'clutterdness' of the environment; a cluttered office environment should have a higher a-priori occupancy probability than, say, an outdoor scene. Indeed it is noted in [6] that the priors "... can be obtained from experimental measurements from the areas of interest, or derived from other considerations about likelihoods of cell states." However, typically one does not bother and the priors are set to the non-informative

$$P(\mathrm{OCC}(c_j)) = P(\mathrm{EMP}(c_j)) = \frac{1}{2}.$$

The prior sensor measurement probability. Similar considerations hold for the prior sensor measurement probability $P(\boldsymbol{s})$. Again, the prior

could be estimated from a great number of sensor measurements taken in the area of interest. In this case, the prior is instead calculated from other probabilities which we have already obtained earlier:

$$P(\boldsymbol{s}) = p(\boldsymbol{s} \mid \mathrm{OCC}(c_j))P(\mathrm{OCC}(c_j)) + p(\boldsymbol{s} \mid \mathrm{EMP}(c_j))P(\mathrm{EMP}(c_j)).$$

This is a valid calculation because in the definition of an occupancy grid (see **??**) it was defined that "occupied" is taken to mean "not empty". Thus the states OCC and EMP are exhaustive and $P(\mathrm{OCC}(c_j)) + P(\mathrm{EMP}(c_j)) = 1$. The conditional probability $p(\boldsymbol{s} \mid \mathrm{OCC}(c_j))$ is obtained with (3.2.3); $p(\boldsymbol{s} \mid \mathrm{EMP}(c_j))$ is calculated similarly. In the previous paragraph it was discussed how the priors $P(\mathrm{OCC}(c_j))$ and $P(\mathrm{EMP}(c_j))$ are obtained.

The result. Combining the sensor model $p(\boldsymbol{s} \mid \mathrm{OCC}(c_j))$ with the priors $P(\mathrm{OCC}(c_j))$ and $P(\boldsymbol{s})$ Elfes arrives at a conversion function as sketched in figure 3.1. For ease of discussion, we call the part with occupancy values

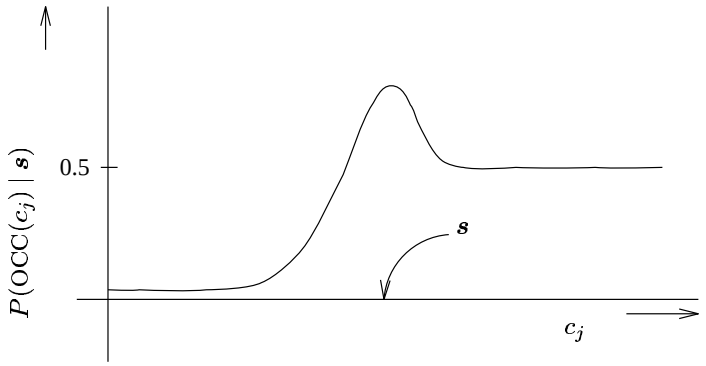

Figure 3.1: Example of basic conversion function for occupancy grids in 1D. For cells closer to the sensor than the measured distance $\boldsymbol{s}$ the occupancy probability is set to 0 (empty). The cells around the measured distance have high occupancy probability. Beyond this range there is no information (and thus the occupancy probability is fixed to the prior $P(\mathrm{OCC}(c_j)) = \frac{1}{2}$.).

close to 0 the *empty part*, or, as in [9], the *free space hypothesis*. This part represents the cells c_k in equation (3.2.2). We will call the part with higher occupancy probabilities the *occupied part*.

In 2D the method results in a conversion function as sketched in figure 3.2[1].

[1]Let it be noted that although similar in form these distributions are the sensor models, while the figures in section **??** are the inverse sensor models.

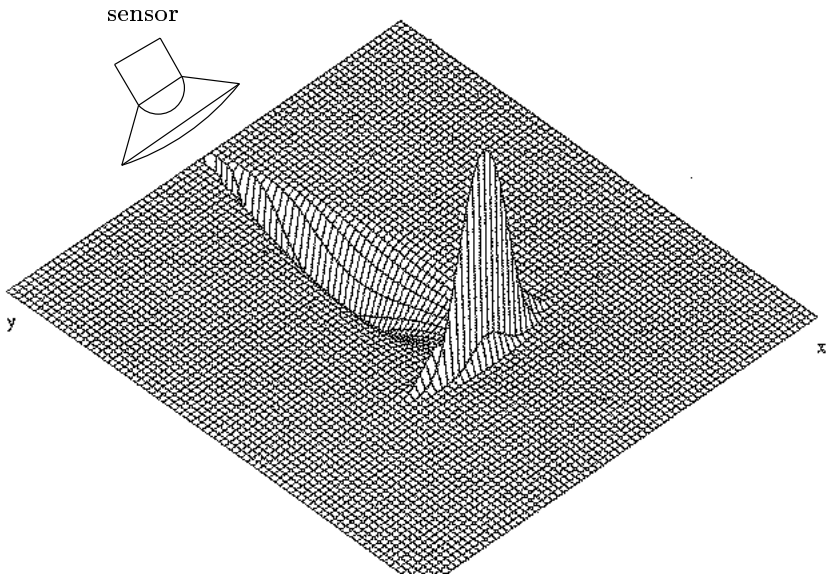

Figure 3.2: Example of basic conversion function for occupancy grids in 2D. The sensor is positioned in the top left of the figure and is pointed towards the bottom right. Again the figure clearly shows the free space hypothesis, the occupied part and the part where no information is available.

Discussion. Our major objection to this approach is that in the calculation of the sensor model in (3.2.3) the distribution of grid configurations $G^{\mathrm{OCC}(c_j)}$ is derived from the individual cell occupancy probabilities. The rationale behind this derivation is that the individual cell occupancy values are assumed to be independent. In our opinion, since the assumption of independence is violated anyway, this is the place to drop that assumption.

The violation of the independence assumption can be seen as follows. Independence between cells in the grid is assumed because the occupied cells are assumed to be occupied by infinitesimal objects. Thus there are no obstacles which cover multiple cells, for in this case the occupancy values in these cells would indeed be dependent. This assumption is justified by the fact that range sensors only measure the distance to single points of an object only; these points are modeled as infinitesimal objects in the grid. However, if these assumptions are carried through in the conversion of sensor measurements, the occupancy values in the resulting grid are no longer independent. Consider, e.g. figure 3.2. The occupied part of the figure represents the probability distribution of the cells being occupied by the *one single* infinitesimal object that was measured by the sensor (in [3, 4] Elfes calls this the "somewhere occupied" region). Thus, if we find that one of these cells does not contain the object (for instance, if another sensor gives occupancy value 0 for that cell) the occupancy values of the other cells in the occupied part can be increased and therefore, they are indeed dependent. This is, in fact, exactly what is done in [5]. In that article an update of an occupancy grid is performed in two steps. In the first step the empty parts in the grid are combined. Then the occupied part is rescaled to probability mass 1 before the remainder is fused.

In our opinion, if the assumption of independence is violated anyway, it should be abandoned at an earlier stage. It is clear that in most "areas of interest" of the robot obstacles will be present that cover multiple cells in the occupancy grid, which are, therefore, dependent. We propose to already use this dependency in the calculation of the sensor model. In the approach presented in this section, this means that the distribution of grid configurations in (3.2.3) is taken over grids with dependent cell values. In our approach, we will take the distribution of grid configurations from the actual environment the robot is working in. This distribution will then be reflected in the grids representing the converted sensor measurements.

This way, in our approach we acknowledge the higher order of a discrete probabilistic representation of the environment, but we choose to represent the zeroth order approximation only. We are freed from the computational burden of also updating the higher orders in our representation of the environment, while we do use the dependencies at will, e.g. in the conversion

of sensor data.

3.2.2 Variations to the basic conversion functions

Many different conversion functions have been presented in literature. Unfortunately, a comprehensive overview of the different methods is still lacking. Therefore we decided to compose one ourselves. To not get distracted from the main topic in this chapter, we have included this overview as an appendix **??**, and we only give an impression of the different approaches here.

One basic problem with the use of Gaussian shaped sensor models is how the parameter of this model, i.e. the width of the Gaussian, should be chosen. Therefore in [1] and in [14] approaches are presented where this width is estimated by experiment. But there are several objections to be made to the assumption of Gaussianity per se.

In some cases rather than Gaussian distributed errors only a strict bound can be given on the error. In such cases the set membership approach can be used (see section **??**, chapter **??**). In other cases the predominant errors may be the so called *outliers*, or *spurious* sensor measurements. One can then follow the approach presented in [13]. In this work a more heavily tailed distribution is used, and a special fusion method is introduced for such distributions. Converse to spurious readings (measurements from obstacles which aren't there), a sensor may also suffer from specular reflections, when a sensor does not measure an obstacle which is there. In [9] a conversion method is introduced which adapts the conversion function to probable specular reflections: if a specular reflection is suspected the empty part of the converted measurement (see figure 3.1) is decreased. Obviously, while these approaches work for the specific type of errors they are designed for, they are not adaptive to other kinds of errors.

Another solution to the problem of non-Gaussian distributed noise is to filter the non-Gaussian component before performing the conversion with standard Gaussian methods. E.g., in [2] polynomial error series are estimated in a calibration phase. The model thus obtained is then used as a "whitening filter" during operation. In [16, 17] a pulse reflected by one transmitter is sensed by three receivers. It is shown that with straightforward triangulation the non-Gaussian uniform error component can be effectively filtered from the signal. Unfortunately, both approaches are very sensor specific and a more general approach is needed.

3.2.3 Learning to convert sensor measurements

Basically, our objections to the conversion methods proposed in the previous section can be stated as follows.

- There is sufficient support in literature that the sensor models should *not* use Gaussian-shaped noise models. The alternatives presented in literature are either noise-specific or sensor-specific. A more general approach is needed.

- In the conversion of sensor measurements of a common internal representation the distribution of the actual 'areas of interest' of the robot should be taken into account. For the environment also has a significant influence on the correct conversion of sensor measurements: e.g., the clutterdness of the environment, the reflection coefficients of the obstacles in the environment (mirrors vs. brick walls), and the air temperature. An approach is needed which adapts to changes in the characteristics of such environments.

  This point was already mentioned in the discussion of section 3.2.1. It implies that the conversion method also takes into account the dependency between cells in the grid caused by the occurence of non-infinitesemal obstacles in the environment.

Summarizing, we are faced with the task to specify very *complex* functions, the conversion function $\mathcal{G}_j$, which are *adaptive* to changes in the environment. In recent years *neural network* theory is often applied to *learn* such functions. We will first shortly review the literature on neural network approaches to the conversion of sensor data and we will introduce our approach in the next section.

3.2.4 Review of learned conversion functions

The idea of using neural networks for the conversion of sensor data is also described by Thrun in [22, 23], although it is applied in a different kind of sensor data fusion system. In the approach of Thrun a neural network is used to learn the conversion of robot-mounted range sensors to a *world*-centered (global) occupancy grid. The network serves as a general function approximator of the conversion function from its inputs, in this case the sensor measurements, to its outputs, in this case the occupancy values for the cells in the global grid.

A single network is used to compute the occupancy values for all cells c_j in the grid individually. At each computation of an occupancy value,

the network is input with the polar coordinates of the cell c_j in the world-centered grid with respect to the current robot configuration as well as the quadruplet of sensor measurements from the four sensors pointed in the direction of that cell. The network outputs the occupancy value g_j of that cell. The network is sketched in figure 3.3.

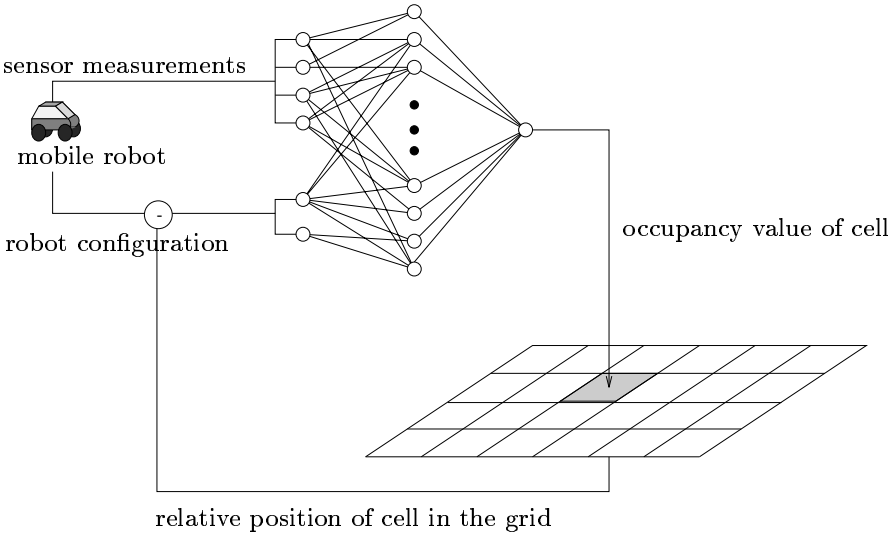

Figure 3.3: Sketch of the network used by Thrun to convert sensor measurements.

The neural network is trained to perform the correct conversion by minimizing the sum squared error over a number of *learning samples* (in section 3.3.3 it is discussed in more detail how such neural networks are trained). A learning sample consists of a pair of input values and the correct, or 'desired' output values. In the work of Thrun, the robot obtains

the training samples by driving around in a so called 'calibration environment'. The occupancy grid representation of this environment is calculated manually prior to the training of the network, thus providing the desired outputs of the network. The robot then drives around in this environment in the training phase while gathering the inputs for the training samples.

Results show that after this training phase, the robot has successfully learned to convert its sensor measurements to the world centered grid. If it is placed in an a-priori unknown environment of similar characteristics, the neural network can form a rather accurate occupancy grid representation of this environment[2].

While this approach does indeed solve our first objection formulated in the previous section, the approach is not adaptive to a change of environment. If the robot is to be operated in a different 'area of interest', first a calibration environment with similar characteristics is needed (indeed, this point is explicitly mentioned in [23]). Obviously it is impractical to create calibration environments for, e.g., space applications or for robots that will be mass-produced to be sold and applied in many different environments.

Furthermore, the network described here may also suffer from the following:

- Because the network converts quadruplets of sensor measurements in fact it performs cooperative fusion between these measurements. We have already stated that cooperative fusion is sensitive to sensor malfunction and thus so is this network. It cannot adapt to a malfunctioning sensor during operation.

- A single network is used to convert quadruplets of many different sensors. Thus the sensor model for this conversion function is averaged over all the sensors on the robot. This will have negative effects when one sensor is much less accurate than the others (during calibration). It will have drastic effects if one sensor malfunctions during calibration. The network does not identify which.

- The network is fed with 6 inputs. Such a high-dimensional input space requires many training samples to learn a reliable conversion. Typically the increase of the number of inputs requires an exponential increase of learning samples, while an increase in the number of networks used requires a linear increase of training samples.

[2]For ease of discussion we have ignored the robot positioning problem here. This problem is addressed, however, in the references given and a rather successful position estimation algorithm is described.

In the next section we introduce a neural network approach to the conversion of sensor data which is, in the first place, adaptive to changes in the environmental characteristics, at the cost of a few robot collisions. Also, we use our networks to convert single sensor measurements. Thus, it is less sensitive to sensor malfunction and it may be expected that it does not need as many learning samples.

3.3 Neural networks for conversion functions

In the introductory section conversion functions $\mathcal{G}_j$ were defined which perform the conversion of sensor measurements to occupancy values (probabilities) in the occupancy grids. We defined the conversion functions as follows (see (3.1.1)):

$$g_j \triangleq P(\text{cell } c_j \text{ is occupied} \mid \boldsymbol{s}) = \mathcal{G}_j(\boldsymbol{s}).$$

Again note that the actual occupancy value of cell c_j is denoted by g_j.

In the previous section it was shown how such conversion functions could be derived from the sensor model $P(\boldsymbol{s} \mid r)$ using Bayes' rule and a number of variations to the basic conversion method were discussed. We also discussed the advantages of *learning* the conversion functions $\mathcal{G}_j(\boldsymbol{s})$ instead. In this approach, no explicit definition of the sensor model is given. The network learns the entire conversion directly, thus implicitly modeling both the model of the sensor as well as the environmental characteristics. In the application of Bayes' rule these aspects are explicitly modeled in the sensor model $p(\boldsymbol{s} \mid \text{OCC}(c_j))$ and the priors $P(\text{OCC}(c_j))$ and $P(\boldsymbol{s})$, respectively. We also reviewed the approach to learning conversion functions introduced by Thrun.

In this section we introduce our learning approach, which is somewhat different. First of all we learn separate conversion functions for each sensor. This way higher level fusion is performed after conversion and unnecessary cooperative fusion is avoided. Perhaps even more important, we introduce an approach which remains adaptive to changes in either the sensor or the environment. The network keeps adjusting its conversion functions during operation. This is due to the fact that we use the *sense-and-drive* algorithm to provide the learning samples for the network.

We will first define the neural network and the learning rule to train the network in detail. We will then describe the sense-and-drive algorithm and we conclude the section with experimental results and a discussion.

3.3.1 Network topology

A neural network is to be defined which learns the conversion functions:

$$g_j = \mathcal{G}_j(\boldsymbol{s}). \tag{3.3.1}$$

To this end a network is defined as sketched in fig. 3.4:

- an input neuron which is fed with the sensor measurement $\boldsymbol{s}$,

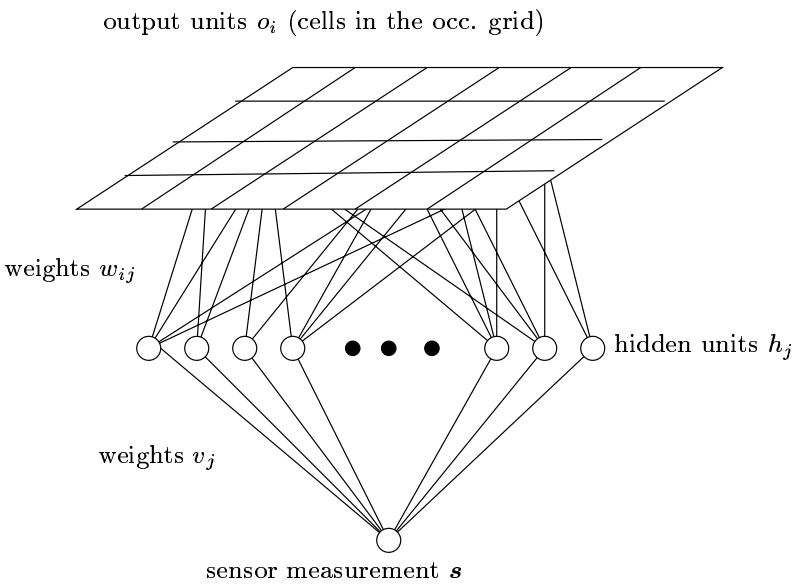

Figure 3.4: The network used for learning the conversion of sensor data.

- a number of hidden units with values h_j, $j = 1, \ldots, N_{\text{hid}}$, with activation function $\mathcal{F}_{\text{hid}}$ and biases Ψ_j,
- a set of output units with outputs $o_i, i = 1, \ldots, N_{\text{out}}$, witch activation function $\mathcal{F}_{\text{out}}$ and biases Θ_i,
- a set of weights v_j, which connect the hidden units to the input unit,
- a set of weights w_{ij}; each output unit i is connected to all hidden units j.

In this application each output unit represents a single cell c_i in the occupancy grid and thus $g_i \equiv o_i$ and $N_{\text{out}} \equiv N$.

3.3.2 Functionality of the network

The output of a single output unit in this network can be written as

$$o_i = \mathcal{F}_{\text{out}} \left(\sum_j w_{ij} \cdot \mathcal{F}_{\text{hid}} \left[v_j \cdot \boldsymbol{s} + \Psi_j \right] \right) + \Theta_i. \tag{3.3.2}$$

We are now faced with the task to adapt the weights and biases in the network such that the difference between o_i and $\mathcal{G}_i(\boldsymbol{s})$

$$\| o_i - \mathcal{G}_i(\boldsymbol{s}) \| \tag{3.3.3}$$

is minimized.

3.3.3 The learning rule

A network as defined here is usually trained with a number of training samples consisting of inputs $\boldsymbol{s}$ and the corresponding 'correct', or 'desired' outputs $G^* = \{g_1^*, \ldots, g_N^*\}$. In the current context the input is the sensor measurement $\boldsymbol{s}$ and the desired grid G^* is the probabilistic representation of the distribution of all possible grid configurations which would give that particular sensor measurement $\boldsymbol{s}$. As we stated earlier, we will use a distribution of grid configurations which is derived from the actual "areas of interest" of the robot.

If we have a set of such samples we could define the *summed squared error* ε of the network over this set as:

$$\varepsilon = \sum_{\text{set}} \sum_i (g_i^* - o_i)^2.$$

In fact, this is the distance between the the output of the network and the desired output as defined in (3.3.3) if you take the Euclidean distance norm. The network can then be trained by minimizing this error term. Minimization is performed by following the negative gradient of the error term with respect to the weights in the network:

$$\Delta w_{ij} = -\gamma \cdot \frac{\partial \varepsilon}{\partial w_{ij}} = 2\gamma(g_i^* - o_i)\frac{\partial}{\partial w_{ij}} o_i, \tag{3.3.4}$$

where $\frac{\partial}{\partial w_{ij}} o_i$ can be obtained from (3.3.2). This learning rule is called the generalized δ-rule. The weights to the hidden units can be adjusted similarly by first back-propagating the summed squared error to the hidden units and then applying gradient descent to the back-propagated error (see [19]).

3.3.4 The training samples

Training with binary samples

Unfortunately, in practice the learning samples $(\boldsymbol{s}, G^*)$ are not readily available. Instead, we train the network with samples $(\boldsymbol{s}, G^{\text{bin}})$ where G^{bin} is a grid configuration representing a specific environment. As defined earlier in chapter **??**, section **??**, a grid *configuration* is a binary representation of a specific environment, in which we know *for certain* whether or not the cells in the grid are occupied. We take these specific environments from the areas of interest of the robot, such that the distribution of the configurations induces G^*.

Thus, we only have training samples with grids G^{bin} characterized by

$$\forall i \qquad g_i^{\text{bin}} \in \{0, 1\}.$$

These values are not probabilities, but realizations of a stochastic process following the probability distributions g_i^*. I.e.,

$$P(g_i^{\text{bin}} = 1) = g_i^*,$$

or in this case

$$E_{G^{\text{bin}},\ \boldsymbol{s}}[g_i^{\text{bin}}] = g_i^*,$$

where the expected value is taken over all possible grid configurations $G^{\text{bin},\ \boldsymbol{s}}$ which give the same sensor measurement $\boldsymbol{s}$.

But what if we train the network with the samples $(\boldsymbol{s}, G^{\text{bin}})$? It still remains to be proven that with such samples the network learns to output the probabilities G^* given the measurement $\boldsymbol{s}$. In [22, 23] Thrun uses

the very same method to train his networks (in this case the specific grid configuration is an occupancy grid representation of the calibration environment), but for a proof that this method does indeed work the reader is referred to [8]. Below we prove that it actually doesn't matter whether you train the network with samples $(\boldsymbol{s}, G^*)$ or the samples $(\boldsymbol{s}, G^{\text{bin}})$.

Theorem 1 *If a supervised neural network is trained with samples* $(\boldsymbol{s}, G^{\text{bin}})$ *the same result is obtained as when the network is trained with* $(\boldsymbol{s}, G^*)$*, provided that the distribution of grid configurations* G^{bin} *follows* G^**.*

Proof The proof shows that with either type of learning samples, the expected change in weights of the network is the same. This means that for a given network, the weights, being the parameters of the network, may be expected to converge to the same values. Thus it does not matter which type of sample is used.

For the case of samples $(\boldsymbol{s}, G^*)$ we have:

$$E_{(\boldsymbol{s},G^*)}\left[\Delta w_{ij}\right] = E_{\boldsymbol{s}}\left[\Delta w_{ij}\right] =$$

$$= E_{\boldsymbol{s}}\left[2\gamma(g_i^* - o_i)\frac{\partial}{\partial w_{ij}}o_i\right]. \tag{3.3.5}$$

In the first equality we used that G^* is a single-valued function of $\boldsymbol{s}$ (the conversion function!), and in the second equation we simply substituted (3.3.4).

For the case of samples $(\boldsymbol{s}, G^{\text{bin}})$ we have:

$$E_{(\boldsymbol{s},G^{\text{bin}})}\left[\Delta w_{ij}\right] = \int_{(\boldsymbol{s},G^{\text{bin}})}\Big(\Delta w_{ij}p(\boldsymbol{s}, G^{\text{bin}})\Big)\,d\boldsymbol{s}dG^{\text{bin}} =$$

$$= \int_{(\boldsymbol{s},G^{\text{bin}})}\Big(\Delta w_{ij}p(G^{\text{bin}} \mid \boldsymbol{s})p(\boldsymbol{s})\Big)\,d\boldsymbol{s}dG^{\text{bin}} =$$

$$= \int_{\boldsymbol{s}}\left(\sum_{G^{\text{bin}},\ \boldsymbol{s}}\Big\{\Delta w_{ij}p(G^{\text{bin}} \mid \boldsymbol{s})\Big\}\right)p(\boldsymbol{s})d\boldsymbol{s} =$$

$$= E_{\boldsymbol{s}}\left[\sum_{G^{\text{bin}},\ \boldsymbol{s}}\Big\{\Delta w_{ij}p(G^{\text{bin}} \mid \boldsymbol{s})\Big\}\right]. \tag{3.3.6}$$

We then simplify the following:

$$\sum_{G^{\text{bin}},\ \boldsymbol{s}}\Big\{\Delta w_{ij}p(G^{\text{bin}} \mid \boldsymbol{s})\Big\} = 2\gamma\sum_{G^{\text{bin}},\ \boldsymbol{s}}\left\{(g_i^{\text{bin}} - o_i)\frac{\partial}{\partial w_{ij}}o_i\ p(G^{\text{bin}} \mid \boldsymbol{s})\right\} =$$

$$= 2\gamma\left\{(1 - o_i)\frac{\partial}{\partial w_{ij}}o_i \cdot g_i^* + (0 - o_i)\frac{\partial}{\partial w_{ij}}o_i \cdot (1 - g_i^*)\right\}.$$

Here we used the fact the G^{bin} follows the distribution of G^*: with probability g_i^* we have that cell c_i is occupied and thus $g_i^{\text{bin}} = 1$, and with probability $(1 - g_i^*)$ we have $g_i^{\text{bin}} = 0$. We further simplify to:

$$2\gamma \left\{ g_i^* \frac{\partial}{\partial w_{ij}} o_i - g_i^* o_i \frac{\partial}{\partial w_{ij}} o_i - o_i \frac{\partial}{\partial w_{ij}} o_i + g_i^* o_i \frac{\partial}{\partial w_{ij}} o_i \right\} =$$

$$2\gamma \left\{ g_i^* \frac{\partial}{\partial w_{ij}} o_i - o_i \frac{\partial}{\partial w_{ij}} o_i \right\} =$$

$$2\gamma (g_i^* - o_i) \frac{\partial}{\partial w_{ij}} o_i. \tag{3.3.7}$$

Finally, if we substitute (3.3.7) in (3.3.6) we get (3.3.5), which completes our proof. □

In this proof we have used the fact that if we sum over all grid configurations G^{bin} for a given $\boldsymbol{s}$, all the parameters in the network, such as o_i and $\frac{\partial}{\partial w_{ij}} o_i$ are constants, because the input $\boldsymbol{s}$ is constant.

Convergence

In conventional supervised learning the network would be presented with training samples $(\boldsymbol{s}, \mathcal{G}_i(\boldsymbol{s}))$. But since such samples are not available the network is trained with samples $(\boldsymbol{s}, g_i^{\text{bin}})$. We just showed that the expected output value of the network $E[o_i]$ equals the expected occupancy value of the cell $E[g_i^{\text{bin}}] = g_i^*$. However, because of this type of training sample the network will never converge.

If we consider the learning rule (3.3.4) with g_i^{bin} substituted for g_i^* we see that the change in the weights never decreases to 0. This is because the output of the network is $o_i \in \mathcal{R}$ while the 'desired output' in the training sample $g_i^{\text{bin}} \in \{0, 1\}$. And thus the network will not converge to an equilibrium state. Instead it will oscillate around this optimum. Convergence can be achieved by averaging the weight update (3.3.4) over a sufficient number of training samples. In our experiments we did not bother and we were satisfied with a function oscillating slightly around its optimum; we stopped training after a fixed number of learning samples.

Partial training: use local activation functions.

Another point which deserves attention is the fact that the samples are most probably not evenly distributed over the inputs $\boldsymbol{s}$. The distribution depends on the actual sensor measurements taken by the robot which are, in turn, dependent on the environment and the exploration strategy.

Because of this uneven distribution of samples over the outputs as well as the inputs, the network should use *local* activation functions for the hidden units. For else the uneven distribution would have a global effect on the learned function. The convergence function would be learned accurately for inputs $\boldsymbol{s}$ which occur frequently, while for inputs $\boldsymbol{s}$ which are infrequent a much worse approximation is learned. Therefore we choose Gaussian kernels for our activation functions:

$$h_j = \exp - (v_j \boldsymbol{s} + \Psi_j)^2.$$

Such networks are more commonly known as Radial Basis Function networks.

3.3.5 The Sense-and-drive algorithm

In contrast to the application of Thrun, where a "calibration environment" was needed, we assume that such environments are not available. Instead, learning samples $(\boldsymbol{s}, G^{\text{bin}})$ are obtained by the robot itself. The robot collects the training samples while driving around in its environment. We call this algorithm for collecting training samples the **sense-and-drive** algorithm:

- With the robot in configuration $\boldsymbol{\phi}$ take a sensor measurement $\boldsymbol{s}$.
- Drive in a random direction until collision occurs.
- Using the odometry of the robot, represent the path just traveled by the robot in an occupancy grid centered at $\boldsymbol{\phi}$. Set $g_i^{\text{bin}} = 1$ for the cell c_i in which collision occurred and set $g_j^{\text{bin}} = 0$ for the cells c_j that were traversed without collision.

An example of the sense-and-drive algorithm is sketched in figure 3.5.

This way, the sense-and-drive algorithm first provides the network with an input $\boldsymbol{s}$ and then the robot starts driving. When the robot is driving, the probability that it will collide with an obstacle in cell c_j by definition is given by $P(\text{cell } c_j \text{ is occupied} \mid \boldsymbol{s})$, and thus

$$P(g_i^{\text{bin}} = 1) = P(c_j \text{ is occupied} \mid \boldsymbol{s}),$$

or

$$E[g_i^{\text{bin}}] = P(c_j \text{ is occupied} \mid \boldsymbol{s}).$$

This is exactly what we need for the training samples as discussed in section 3.3.4.

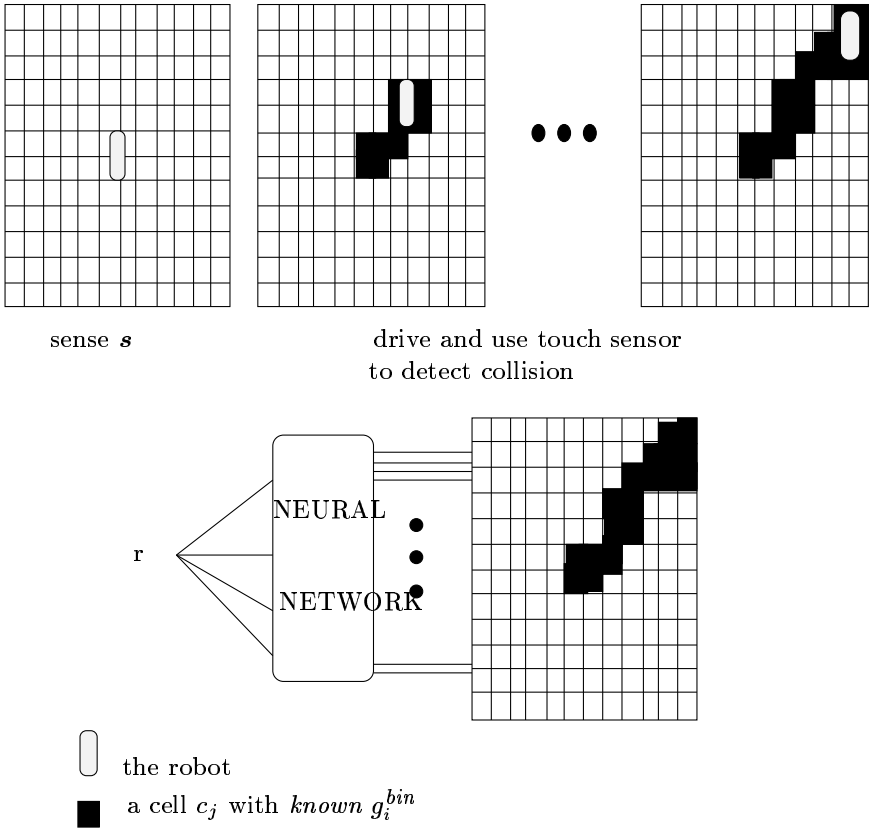

Figure 3.5: Illustration of how training samples $(\boldsymbol{s}, G^{\text{bin}})$ are obtained.

Sense-and-drive at run-time. The successful application of the sense-and-drive algorithm requires frequent collision of the robot. Obviously, this cannot be realized at run time and this specific algorithm can only be applied in a calibration phase. However, at run time the networks which perform the conversion of sensor data remain adaptive: the network can be trained with training samples $(\boldsymbol{s}, G^*)$ where $\boldsymbol{s}$ are the measurements taken by the robot at run time and where G^* are the occupancy grids representing the robot's local environment, obtained by the fusion of the various converted sensor measurements. I.e., the "desired" output is in this case given by the fused outputs of all the sensors. This idea is also proposed in the next chapter for a different kind of network. It is discussed in more detail in chapter **??**.

In case of sensor malfunction, the robot may be provided with erroneous occupancy grids of its local environment. In the worst case this will result in a collision with the obstacles in the environment. However, the sense-and-drive algorithm exploits the occurrence of this collision to adapt the conversion function of the malfunctioning sensor. Thus, at the cost of a few possible collisions, the algorithm is capable of adapting its inverse sensor model.

3.4 Experiments in simulation

In this section we discuss various experiments to learn sensor models in a simulated robot environment. The learning algorithm described in the previous section was tested with a simulator of an ultrasonic range sensor (ASSIM, see [10]). After the robot takes a sensor measurement $\boldsymbol{s}$ with the simulated acoustic sensor, it follows a path to explore the (local) environment with touch sensors. During this path more sensor measurements are taken and stored to generate training samples. The path is then represented in an occupancy grid G^{bin} and the network is trained with $(\boldsymbol{s}, G^{\text{bin}})$. The path consists of a series of connected straight lines, each at random angle. If a collision occurs, the path is abandoned and a new path is started at a random location. In figure 3.6 the environment, the robot, and the size of the robot-centered occupancy grid are sketched on scale.

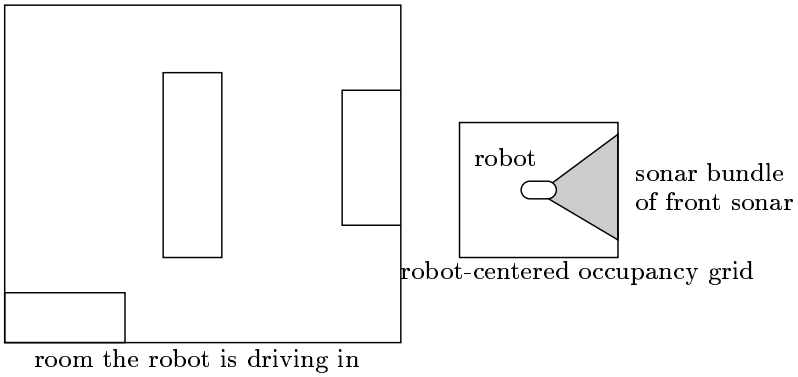

Figure 3.6: Sketch (on scale) of the learning environment.

In our experiments we used a network as defined in section 3.3.1 with one input neuron, $N_{\text{hid}} = 20$, $N_{\text{out}} = 16 \times 16$, where the outputs neurons are organized in a regular 2D square lattice. For the activation function $\mathcal{F}_{\text{out}}$ we used the identity function and for $\mathcal{F}_{\text{hid}}$ we used Gaussian functions. The basis functions Ψ_j were evenly distributed over the input range. The widths v_j were initialized at half of the distance between two adjacent kernels and were updated during training using error back-propagation and the generalized δ-rule. The biases Θ_i were kept fixed at 0.5, representing 'unknown' occupancy.

3.4.1 A model for a single sensor

As a first experiment, we learned a conversion for a single sensor mounted on the front of the robot, as shown in figure 3.6. In figure 3.7 results are shown of the learned sensor model. The results clearly show the increasing width of the sonar bundle at higher range measurements. This bundle is also wider than one may have expected because the sonar sensor is a point-source on the center of the robot, while the touch sensors are mounted on the sides of the robot. Thus, if an obstacle is measured in front, a collision may very well occur more to either of the sides of the robot. This shows that the network also learns to incorporate the size of the robot in the sensor model.

Furthermore, a comparison of these models with figure 3.2 shows that the learned sensor model represents a much larger free space hypothesis. It is important to note that figure 3.2 only displays that part of occupancy grid which is in front of the sensor, while figure 3.7 displays the entire robot-centered occupancy grid, also including the parts to the left, to the right and behind the sensor. Thus the free space in figure 3.7 also extends to parts of the grid where the sensor is not even pointed at. This can be interpreted as follows: if the sensor measures an obstacle in front of the robot the probability of an obstacle being present behind the robot is almost 0. This is consistent with the training samples presented to the network: if an obstacle is in front of the robot, there is low probability of an obstacle being present at its rear, i.e. the robot does not often drive in corridors. This is an example of how the network incorporates 'knowledge' of the environment. In the following experiment we show how a different model is obtained for a different environment.

3.4.2 A model for a different environment

In this experiment we trained the same sensor as before, but in a different environment containing many more corridors. The new environment is sketched in figure 3.8. Because this environment does contain many corridors, we would expect a different free-space hypothesis for the robot. In this environment, if the sensor measures the presence of an obstacle in front of the robot, it is no longer the case that the cells in the grid that lay behind the robot are probably empty. This is reflected in figure 3.9. Indeed the main difference between this model and the model reflected in figure 3.7 is the free space hypothesis. In this sensor model, if the sensor measures an obstacle in front of the robot it concludes there is also high probability of an obstacle being present behind the robot.

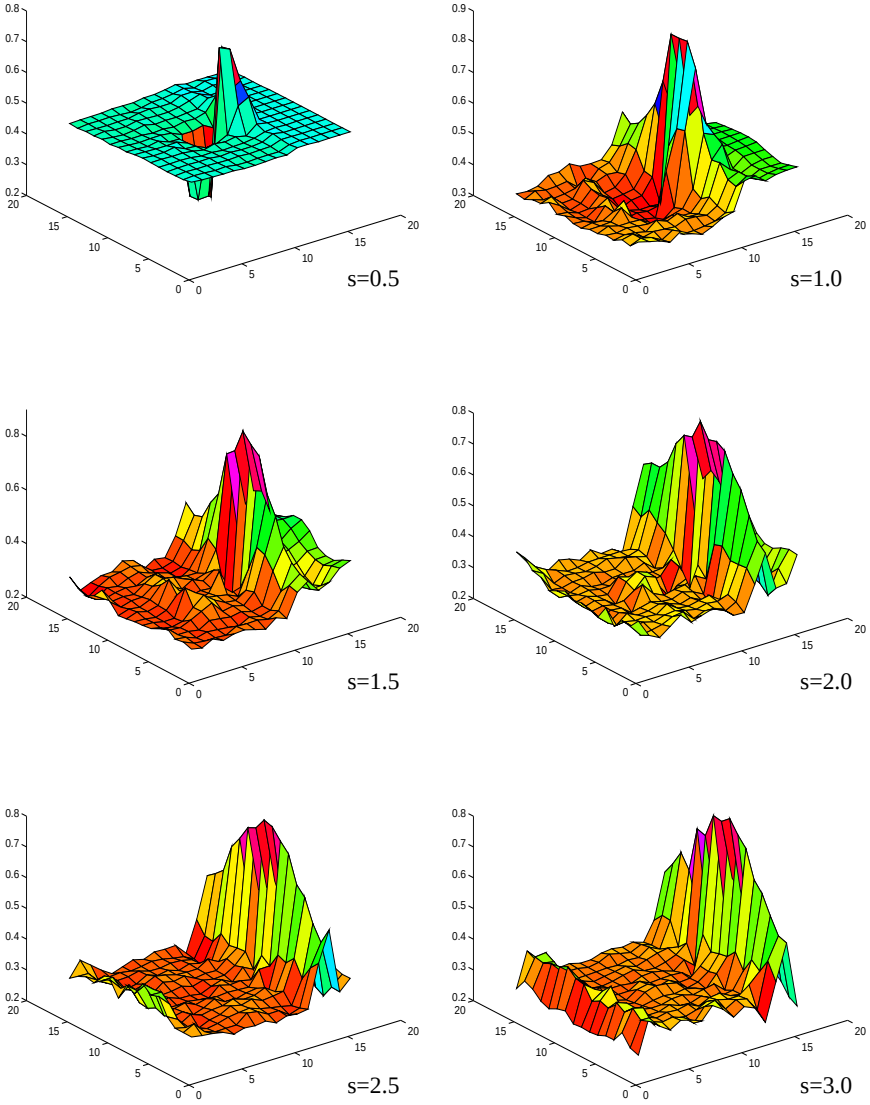

Figure 3.7: The learned sensor model for the acoustic sensor. Shown are the occupancy values $P(\text{cell } i \text{ not empty} \mid \boldsymbol{s})$ in the robot-centered grid for various range measurements $\boldsymbol{s}$. The position of the sensor is in the middle of the grid at (8,8) and it is aimed toward 'the front' of the robot, corresponding to towards the top right of the figure.

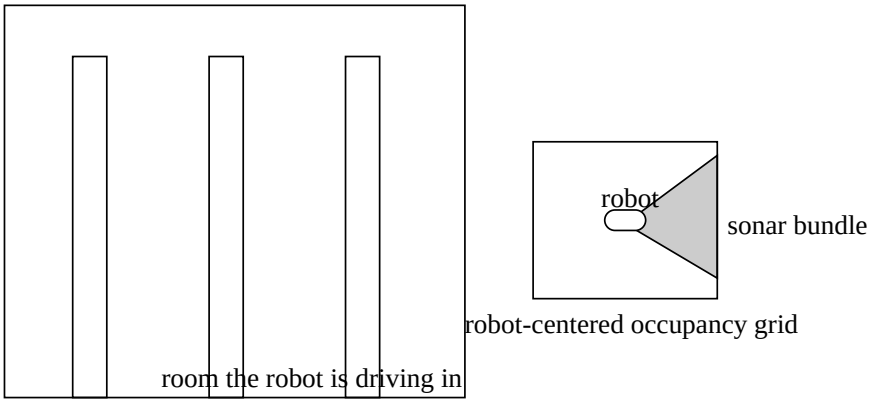

Figure 3.8: Sketch (on scale) of another learning environment containing many more corridors.

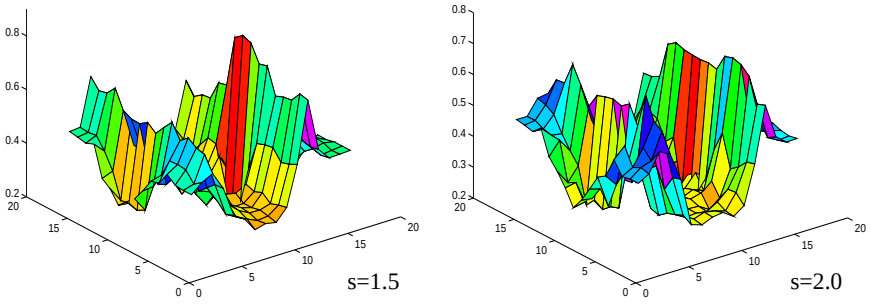

Figure 3.9: A sensor model for the same sensor as in figure 3.7 trained in a different environment containing many corridors.

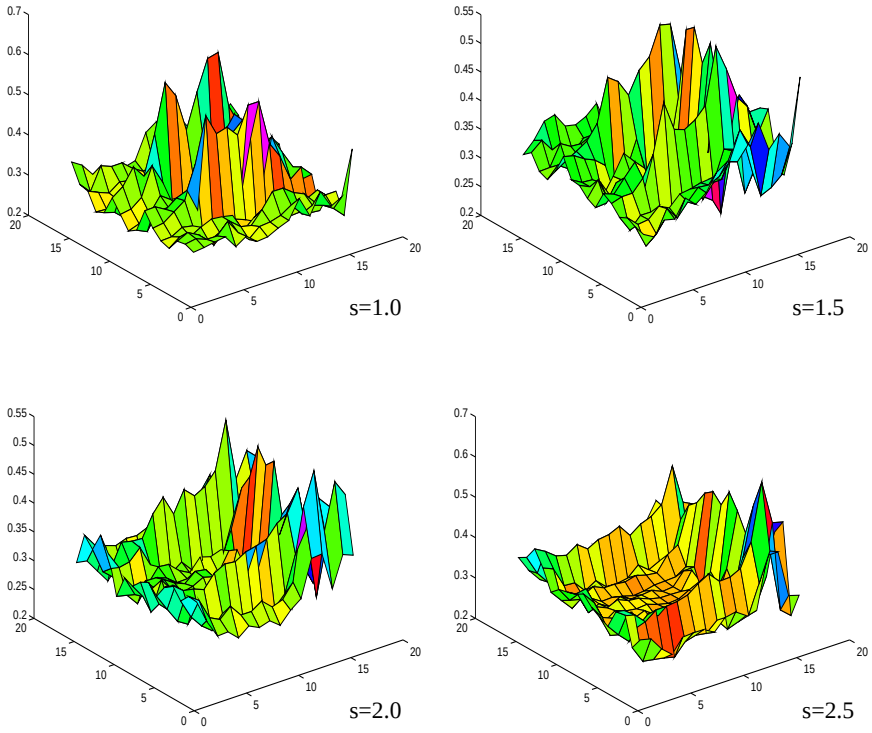

Figure 3.10: A sensor model trained in the same environment as in figure 3.7 for a modified sensor with much larger opening angle.

3.4.3 A model for a different sensor

In another experiment we trained the sensor model in the same environment as sketched in figure 3.6 but with a modified sensor. More specifically, we increased the opening angle of the sensor to 130 degrees[3]. This means that if a sensor measures the presence of an obstacle, this obstacle may very well be positioned much more to the sides of the robot. The results in figure 3.10 clearly show that our neural network also adapts to this change in the sensor parameters. When the sensor still pointing towards the front of the robot measures a certain distance, the model finds out that at this distance an obstacle may as well be positioned at the side as in front of

[3] An impractical opening angle, but useful for demonstration purposes.

the robot. Note that overall the occupancy probabilities are lower: when the sensor pointing towards the front of the robot measures an obstacle the space in front may very well be empty (because the reflection came from the sides).

3.4.4 A model for a different robot

Show that a different model is obtained if a tiny robot is used instead of the large MARIE robot.

3.5 Experiments with a real robot

Discuss experiments with MARIE on sensor models only. Just give some pictures of the models obtained with the data from Marie and discuss.

3.6 Pairwise dependent sources

So far we have only been concerned with the conversion of *single* sensor measurements $\boldsymbol{s}$ to the robot centered occupancy grid. But consider again the higher level fusion method that we propose to use in our SDF system:

$$\prod_{i=1}^{n} \left[\frac{P(z \mid \boldsymbol{s}_i \boldsymbol{s}_{i+1})}{P(z \mid \boldsymbol{s}_{i+1})} \right] P(z)$$

It can be seen that this higher level fusion method does not only require conversion of sensor measurements $P(\mathrm{OCC} \mid \boldsymbol{s})$ but also the conversion of *pairwise dependent* measurements $P(\mathrm{OCC} \mid \boldsymbol{s}_i, \boldsymbol{s}_{i+1})$. Conversion of pairwise dependent measurements requires computation of $P(z \mid \boldsymbol{s}_i \boldsymbol{s}_{i+1})$ given the dependent measurements $\boldsymbol{s}_i$ and $\boldsymbol{s}_{i+1}$. With traditional conversion methods this would require the definition of a sensor model

$$P(\boldsymbol{s}_i \boldsymbol{s}_{i+1} \mid \mathrm{OCC}(c_j)).$$

Obviously such models are rather complex and this is probably why independence is often assumed with traditional conversion methods. But our learning method is easily extendible to learning the conversion of pairwise dependent sources. The neural network is simply extended with an additional input unit and at the hidden layer an extra dimension of hidden units is added. Results of experiments with such networks are presented below.

3.6.1 Simulation results

More specifically, the network for converting pairwise dependent sources is extended as follows (with respect to the definition given in section 3.3.1):

- an additional input unit is added,
- the hidden layer is extended to a 2D grid of 10×10 units.

The basis functions Ψ_j are distributed evenly over the input space.

We also experimented with the training of a model for such pairwise dependent sources. To this end an inverse sensor model was learned for two adjacent sensors pointed towards the side of the robot. The main axes of the sensors lay 15% apart. In figure 3.11 the positions of the sensors are sketched. The results are shown in fig. 3.12. Note that in the bottom two figures the z-axes rescaled for display purposes. The figure clearly shows the model was trained in the environment without corridors. For correlated sensor measurements (the top two grids in the figure) the model gives high occupancy probabilities for a long stretch of cells to the side of the robot.

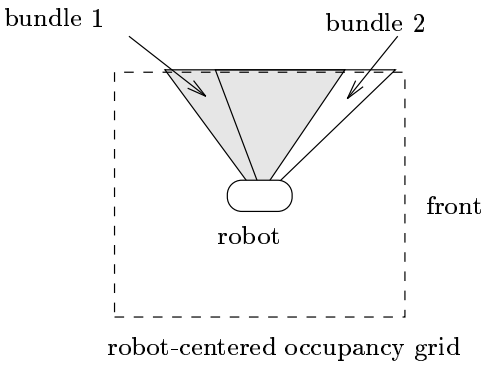

Figure 3.11: Positions of the pairwise dependent sensors for which the conversion function is learned.

For uncorrelated measurements (bottom two figures) the model is uncertain and gives "unknown" occupancy probabilities around 0.5.

This is an important result which indicates the system's behavior when spurious readings occur or when one of the sensor malfunctions. In such cases, the two sensors would 'disagree' and return significantly different results. The figure shows that these different measurements are converted to a basically "unknown" representation of the environment. In other words, the spurious sensor reading is ignored.

3.6.2 Experiments with a real robot

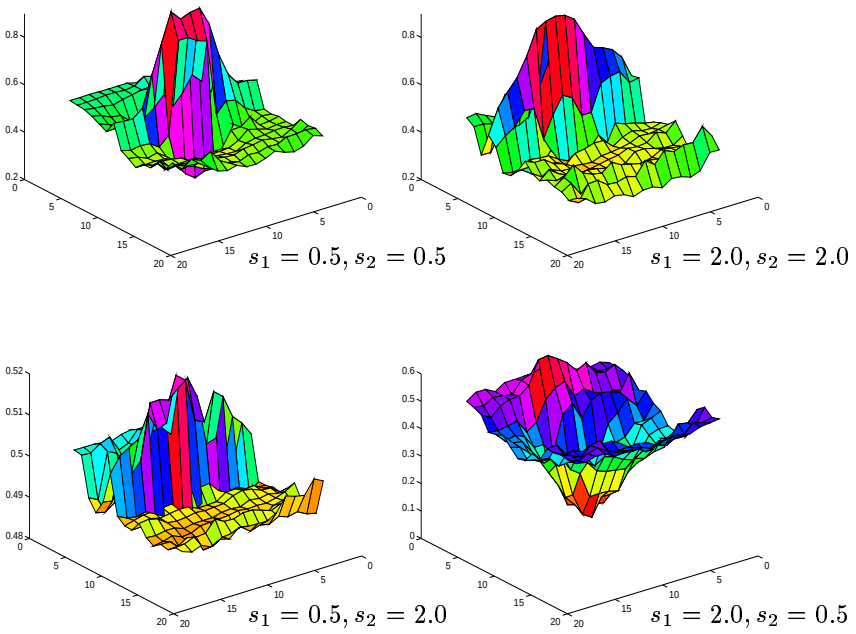

Figure 3.12: The learned inverse sensor model for pairwise dependent acoustic sensors. Shown are the occupancy values grid $P(\text{cell } i \text{ not empty} | \boldsymbol{s}_i \boldsymbol{s}_{i+1})$ in the robot-centered grid for various range measurements $\boldsymbol{s}_i \boldsymbol{s}_{i+1}$. The position of the sensors is in the middle of the grid at (8,8) and they are pointed toward 'the side' of the robot, i.e., towards the top left of the figure.

3.7 Discussion

In this section we have discussed the conversion of sensor measurements $\boldsymbol{s}$ to the probabilities g_j in the robot-centered occupancy grid. We first reviewed the basic approach described by Elfes, in which Bayes' rule is used to perform the conversion. The most important term in this conversion rule is the sensor model $p(\boldsymbol{s} \mid r)$. While Elfes still uses Gaussians as sensor models many variations to this conversion method are concerned with defining more accurate sensor models. These variations were reviewed in section 3.2.2 and appendix **??**. Most of these variations adapted the sensor model to one particular situation, e.g., the occurrence of many spurious readings or the occurrence of many specular reflections. We argued that instead it would be preferable to *learn* the conversion functions with a neural network. In this case the combined effect of the sensor's characteristics as well as the environmental influences on the conversion function would be learned by the network. We reviewed one such approach described by Thrun in which the sensor measurements are converted to a world centered occupancy grid by a neural network. The network was trained using a "calibration environment". We argued that in this particular approach the adaptive potential of neural networks was not fully exploited. Therefore, we introduced our approach, in which sensor measurements are converted to the robot-centered grid. The learning samples are provided by the sense-and-drive algorithm, which remains adaptive during operation.

Unfortunately, it is difficult to compare the different conversion methods, because an objective comparison measure is lacking. There is no cardinality amongst conversion functions. One such comparison measure could, e.g., be the distance between the actual and the predicted occupancy state of a cell:

$$\|P(\mathrm{OCC}(c_j) \mid \boldsymbol{s}) - g_j\|.$$

However, since this is exactly the error mass we are minimizing in our network, our network is superior by definition and thus this measure is of little academic significance.

By definition of the theory on SDF systems we discussed in chapter **??**, the 'goodness' of a particular conversion function is determined by the fitness of the resulting representation for solving the task at hand. In our application this would translate as

> how well can we navigate with the occupancy grids obtained by the conversion of sensor measurements?

But once again this question is difficult to answer in our case. In the work of Elfes and Thrun sensor measurements are converted to a world-centered occupancy grid which is used by a global navigation method. In

our case sensor measurements are converted to a robot-centered representation which is better suited for local navigation methods. Thus, we would effectively be comparing the combined performance of conversion function and navigation method (of which the navigation method is probably the predominant factor).

We have, however, demonstrated that our method is *adaptive* to both the sensor's internal parameters (e.g., the opening angle) as well as environmental influences (e.g., the occurrence of many hallways). This can also be interpreted as a conversion method which can be applied to all different range sensors in all different environments. Rather than specifically adapting a basic model to a special kind of sensor, e.g., which gives many spurious readings, or a specific environment, e.g., where frequent specular reflections occur, we have developed a method which learns the combined effect and which is adaptable to changes in these effects.

In the remainder of this section we discuss the implications of the neural network conversion functions which we introduced in this chapter to the rest of our work presented in earlier chapters. If you will, we tie the loose ends from previous chapters which could not be answered until now.

3.7.1 Supra Bayesian fusion revisited

As discussed in section **??** most Bayesians advocate the use of Supra Bayesian higher level fusion. In the context of our SDF system the application of this fusion system would require the definition of functions f_j which give the probability of cell c_j being occupied given the output o_j of the network which performs the conversion of sensor measurements:

$$f_j(o_j) \triangleq P(\mathrm{OCC}(c_j) \mid o_j). \tag{3.7.1}$$

The rationale behind this higher level fusion method is that you quantify some kind of confidence in the conversion function. E.g., if the conversion function would be very accurate we would have $f_j(o_j) = o_j$, the identity function, while if we have no confidence at all in the conversion function we would have $f_j(o_j) = 0.5$.

In our SDF system we introduced neural networks which learn the conversion functions, and which are adaptive after calibration. Now consider a conversion function in which we have low confidence,

$$\|P(\mathrm{OCC}(c_j) \mid o_j) - o_j\| > \epsilon. \tag{3.7.2}$$

In this case the sense-and-drive algorithm would provide the actual occupancy probability $P(\mathrm{OCC}(c_j) \mid o_j)$ to the neural network which performed

the conversion. And consequently, the conversion function would adapt to this difference. In fact, the neural network is effectively trying to minimize 3.7.2. I.e., by definition the network maximizes the confidence in itself. Thus, we should have maximum confidence in the conversion functions and indeed we should use $f_j = \mathcal{I}$, the identity function.

This way we have shown that if adaptive conversion functions are used, we should have maximum confidence in the converted measurements and consequently, there is no need for a Supra-Bayesian fusion rule. And thus we are justified to use a simpler fusion rule such as the PDOP (as defined in section **??**).

3.7.2 Conversion to Dempster-Shafer grids

In section **??** we discussed Dempster-Shafer grids as an alternative to Bayesian grids. These Dempster-Shafer grids do have some advantages, most noticeably the explicit representation of the 'unknown' state and the less weight that is given to the prior distribution (see [24, 15, 25]). As an argument for probabilistic grids we used the fact that accurate conversion functions for DS grids are still lacking.

The advantage of the 'unknown' state in DS grids can be effectively used in the conversion of sensor measurements. Suppose, e.g., we have a sensor which is accurate for 80% of the time but which returns gibberish elsewhile. The gibberish can, e.g., be spurious readings or specular reflections. In this case a measurement of this sensor would be converted to $\text{OCC}(c_j) = 0.8$, $\text{EMP}(c_j) = 0.0$, $\text{UNK}(c_j) = 0.2$. This means that with certainty 0.8 the cell is occupied, but with certainty 0.2 it is an erroneous reading, in which case it cannot be determined whether the cell is occupied or not (and thus unknown)[4]. Such are the conversion functions which are used in [24, 15]. A rather heuristic inverse sensor model is defined, using the typical opening angle of the sensor, which is then parameterized by this gibberish percentage.

While this is indeed a theoretically more appealing conversion of a sensor measurement, the actual values of the parameters of the conversion (the 'gibberish percentages') are more difficult to obtain. In [24] the standard deviation of the sensor measurements is used to determine this parameter. But as we argued before, because the values of these parameters are influenced not only by the sensor itself but also by the environment the sensor is operating in, the values are preferably *learned* by a neural network. But in the learning method described in this section we have seen that if

[4]The example given here only holds for the "occupied part" of the converted measurement. A similar argument holds for the empty part.

we learn the conversion of sensor measurements, we are actually learning *probabilities* of a cell being occupied (see section 3.3.5). This is one of the main reasons why we choose to work with a probabilistic, rather than a Dempster-Shafer representation.

Bibliography

[1] P. Bessière, E. Dedieu and E. Mazer, "Representing robot/environment interactions using probabilities: the beam in the bin experiment"

[2] Y.D. Chen and J. Ni, "Dynamic calibration and compensation of a 3-D laser radar scanning system", *IEEE Trans. on Rob. and Aut.*, vol 9, no.3, June 1993, pp.318-323.

[3] H.P. Moravec, A. Elfes, "High resolution Maps from wide angle sonar", 1985 IEEE Int. Conf. on robotics and Automation, pp. 116-121.

[4] A. Elfes, "Sonar-based real-world mapping and navigation", IEEE Jorunal of Robotics and automation, Vol. RA-3 no.3, June 1987, pp. 249-265.

[5] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *IEEE Computer*, pp. 46-57, June 1989.

[6] A. Elfes, "Occupancy grids: a stochastic spatial representation for active robot perception", *Proc. of the Sixth Conf. of Uncertainty in AI*, July 1990, pp. 60-70.

[7] E. Fogel, Y.F. Huang, "On the value of information in system identification - bounded noise case", *Automatica* Vol 18., No. 2, 1982.

[8] J. Hertz, A. Krogh, R.G. Palmer, "Introduction to the theory of neural computation", Addison-Wesley Pub. Co., Redwood City, California, 1991.

[9] K. Konolige, "A refined method for occupancy grid interpretation", *Proc. of Workshop Reasoning with Uncertainty in Robotics (RUR)*, Amsterdam, 1995. Proc to appear feb. 1996.

[10] B.J.A. Kröse and E. Dondorp,"A Sensor Simulation System for Mobile Robots", in: T. Kanade, F.C.A. Groen and L.O. Hertzberger (ed.), *Intelligent Autonomous Systems* 2, December 1989.

[11] J. Manyika and H. Durrant-Whyte, "Data fusion and sensor management. A decentralized information-theoretic approach", Ellis Horwood publishers, 1994.

[12] L. Matthies, A. Elfes, "Integration of Sonar and Stereo Range Data Using a Grid-Based Representation", *Proc. of 1988 IEEE Int. Conf. on Rob. and Aut.*, pp 727-733.

[13] M.J. Mirza and K.L. Boyer, "Performance evaluation of a class of M-estimators for surface parameter estimation in noisy range data", IEEE Trans on Robotics and Automation, Vol. 9, no 1, Feb 1993, pp. 75-85.

[14] Moravec and Blackwell, "Learning sensor models for evidence grids", *Robotics Institute Research Review*, Pittsburgh, PA, 1992.

[15] D. Pagac, E.M. Nebot, H.F. Duurant-whyte, "An evidential approach to probabilistic map-building", *Proc. of the 1996 IEEE Int. Conf. on Robotics and Automation*, Minneapolis, April 1996.

[16] H. Peremans, K. Audenaert, J.M. van Campenhout, "A high-resolution sensor based on tri-aural perception", IEEE Trans on Robotics and Automation, Vol. 9, no 1, Feb 1993, pp. 36-48.

[17] H. Peremans, "A maximum likelihood algorithm for solving the correpondence problem in tri-aural perception", *Proc. of 1994 Int. Conf on Multisensor Fusion and Integration for Intelligent Systems*, Las Vegas, Nevada, October 1994, pp. 485-492.

[18] A. Preciado, D. Meizel, A. Segovia, M. Rombaut, "Fusion of multisensor data: a geometric approach", Proc. of the 1991 IEEE Int. Conf. on Robotics and Automation, Sacramento, California, April 1991, pp. 2806-2811

[19] D. E. Rumelhart, G. E. Hinton, R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol 323, pp. 533-536, 1986.

[20] A. Sabater, "Set Membership approach to the propagation of uncertain geometric information", *Proc. of 1991 IEEE Int. Conf. on Rob. and Aut.*, Sacramento, California, April 1991, pp 2718-2723

[21] Y. Shirai, "Visual Sensor Fusion", *Proc. of 1994 Int. Conf on Multisensor Fusion and Integration for Intelligent Systems*, Las Vegas, Nevada, October 1994. Tutorial.

[22] S.B. Thrun, "Exploration and model building in mobile robot domains", Proc. of IEEE Int. Conf. on Neural Networks, San Francisco, CA, March 28 - April 1, 1993.

[23] S.B. Thrun, A. Bücken, "Learning maps for indoor mobile robot navigation", Tech. Report CMU-CS-96-121, School of Computer Science, Carnegie Mellon University, Pittsburgh PA 151213.

[24] A.P. Tirumalai, B.G. Schunck, R.C. Jain, "Evidential reasoning for building environment maps", IEEE Trans. on Systems, Man and Cybernetics, Vol. 25, no 1, Jan 1995, pp 10-20.

[25] F. Voorbraak, artikel moet ik nog zoeken.