Chapter 8

NEURAL FUZZY TECHNIQUES IN SONAR-BASED COLLISION AVOIDANCE

I. Ahrns *
Research & Technology
Daimler-Benz AG
Ulm, Germany

J. Bruske
Computer Science Institute
Christian-Albrechts-
University
Kiel, Germany

G. Hailu
Computer Science Institute
Christian-Albrechts-
University
Kiel, Germany

G. Sommer
Computer Science Institute
Christian-Albrechts-
University
Kiel, Germany

In this chapter we report application of neuro-fuzzy control to sonar based collision avoidance of our TRC labmate robot, Figure 5. To this end, we will first provide the reader with a brief overview of existing concepts of neuro-fuzzy control and then present our own approach based on Radial Basis Functions. This particular Fuzzy-RBF (FRBF) approach is innovative w.r.t. three aspects of neuro-fuzzy control. First, it alleviates the covering problem in fuzzy control, i.e. the problem of an exponential growth of the number of rules with the dimension of the input space. Second, it provides a means for exact interpolation, i.e. inspite of overlapping membership functions the output of the controller can be guaranteed to take the value of the i-th rule if it has degree of fulfillment one. Finally, by using DCS, [1], instead of RBF networks, output calculation of the controller is very fast on average, since only a few rules (the best matching ones) are evaluated on presentation of an input to the controller.

Utilizing FRBF-based controllers we then present two solutions to the collision avoidance problem faced by mobile robots. The first one is a reactive, behavior-based approach in which collision avoidance is implemented as an in-

*The work reported in this chapter was performed at Christian-Albrechts-University at Kiel, Germany

dividual high priority behavior. In such an architecture the behavior selection problem must be solved, i.e. in what situations which behavior has to take over control. Consistent with the fuzzy approach we use a fuzzy blending scheme based on a "pain" function. In our second approach to sonar-based collision avoidance we avoid typical deadlock problems by a closer interaction between higher level tasks and FRBF-based collision avoidance as well as by more advanced feature extraction. Here the task of the controller is to safely follow the freespace direction.

A chapter on neuro fuzzy control would not be complete without a demonstration of its learning capabilities. These are illustrated in Section 4, taking reinforcement learning of collision avoidance as an example.

Since reactive approaches to collision avoidance based on sonar sensors only work if highly erroneous readings caused by crosstalk, bad reflection properties of the environment and shielding problems of the sensors can be eliminated, we finally present a sensor pre-processing method based on sensor grouping and a modified extended Kalman filter algorithm. This easy-to-implement method works very well in practice and is compatible with most existing approaches to fuzzy collision avoidance.

1 Neuro-Fuzzy Control

According to a classification put forward in [2] there are two principal approaches to neuro-fuzzy control. The first one is *cooperative neuro-fuzzy control*, in which the fuzzy controller and the neural network remain seperated, the second one is *hybrid neuro-fuzzy control*, in which the fuzzy controller is realized as a neural network. In cooperative neuro-fuzzy control as e.g. employed in [3] the neural network is used for off-line generation of the membership functions or linguistic rules from training data, typically by clustering the data, or the network is used for online adaptation of membership functions or the weighting of rules in the fuzzy controller.

In this chapter emphasis is on hybrid neuro-fuzzy control. In particular, we exploit the functional equivalence between a restricted class of Sugeno-type fuzzy controllers and Radial Basis Function (RBF) networks as observed in [4] and explained in Section 1.1. Utilizing this equivalence, prior knowledge in form of fuzzy rules can be used to prestructure and initialize an RBF network. The latter can then be trained and refined on training data, and the result of training can reversly be interpreted as fuzzy rules.

While theoretically simple, RBF-based hybrid neuro-fuzzy control has a number of practical problems. These problems are alleviated by extending RBF netwoks in a way described in Section 1.2, resulting in Fuzzy RBF networks (FRBF). The applicability of this approach is demonstrated in Section 2 and 3, and its capability of learning in Section 4.

1.1 On the Equivalence between RBF Networks and Sugeno Type Fuzzy Control

Normalized RBF networks are function approximators ($\mathbb{R}^n \to \mathbb{R}^m$) and calculate their output according to the evaluation function

$$y(x) = \frac{\sum_{i=1}^{N} o_i h_i(x)}{\sum_{i=1}^{N} h_i(x)}, \tag{1}$$

where $h_i(x) = \exp(-\frac{(x-c_i)^2}{\sigma^2})$ denotes a Gaussian radial basis function with center $c_i \in \mathbb{R}^n$. We will refer to the $o_i \in \mathbb{R}^m$ as *output vectors.*

Sugeno-type fuzzy controllers, [2], consist of a set of N linguistic fuzzy rules

$$R_i : \underbrace{\textbf{if} \underbrace{X_1 \text{ is } \mu_{i1}}_{\text{proposition}} \wedge \ldots \wedge X_n \text{ is } \mu_{in}}_{\text{antecedent}} \underbrace{\textbf{then } f_i(x)}_{\text{consequent}}, \tag{2}$$

where $f_i : \mathbb{R}^n \to \mathbb{R}^m$ are functions in x. The output of a Sugeno-type fuzzy controller is computed according to the defuzzyfication formula

$$y(x) = \frac{\sum_{i=1}^{N} f_i(x)\tau_i(x)}{\sum_{i=1}^{N} \tau_i(x)}, \tag{3}$$

where $\tau_i(x)$ is the degree of fulfillment of the i-th antecedent. Now (1) and (3) become identical if

- consequent functions f_i are restricted to constant functions, i.e $f_i(x) = o_i$,
- membership functions $m_{lj}(x)$ are restricted to gaussians, i.e. $m_{lj}(x_l) = \exp(-\frac{(x_l-\mu_{lj})^2}{\sigma^2})$, where $T(X_l) = \{T_{l1}, \ldots, T_{lk_l}\}$ is the linguistic term set of the linguistic variable X_l, $(1 \leq l \leq N)$ with membership functions $M(x_l) = \{m_{l1} \ldots, m_{lk_l}\}$ and
- the fuzzy conjunction is implemented as the algebraic product.

In this case, (3) can be written as

$$y(x) = \frac{\sum_{i=1}^{N} o_i \exp(-\frac{(x-c_i)^2}{\sigma^2})}{\sum_{i=1}^{N} \exp(-\frac{(x-c_i)^2}{\sigma^2})} = \frac{\sum_{i=1}^{N} o_i h_i(x)}{\sum_{i=1}^{N} h_i(x)}, \tag{4}$$

with $c_{ij} = \mu_{ij}$ for proposition "X_j is μ_{ij}" in the antecedent of the i-th rule.

1.2 The Fuzzy-RBF Network

Problems with RBF-based neuro-fuzzy control are that the number of nodes (fuzzy rules) grows exponentially with the dimension of the input space and that due to overlap at the centers the basis (membership) functions interfere with

each other. Another problem is that all rules need to be evaluated, even if they have a very low degree of fulfillment. In the following, we show how exponential growth of the number of rules can be circumvened by incompletely specified antecedents. We also provide a solution to the inference problem. Finally, we show how Dynamic Cell Structures (DCS), [1], help to avoid evaluating all rules by evaluating only the best matching rule and its topologically neighbors.

1.2.1 Incomplete Rules

In high dimensional input spaces an RBF-based controller faces the problem of an exponential explosion of the number of nodes (rules) if the input space has to be uniformly covered. If the input dimension is n and we have a rule depending on only k variables, this rule has to be expanded to l^{n-k} rules if each variable takes l linguistic values. This problem can be alleviated if we drop the requirement that each node in the network must compute an activation function in n dimensions. Instead we must allow for nodes computing an activation function in just k dimensions, if k is the number of input variables in the linguistic rule. For that purpose, we introduce the additional symbol $\perp$, denoting an undefined value, and set c_{ik}, the k-th component of the i-th center to

$$c_{ik} = \begin{cases} \mu_{ik} & : \text{ proposition } X_k \text{ is } \mu \text{ belongs to the antecedent of the } i\text{-th rule} \\ \perp & : X_k \text{ does not appear in the antecedent of the } i\text{-th rule} \end{cases} . \tag{5}$$

The activation function corresponding to that antecedent is calculated as

$$h_i(x) = \exp(- \sum_{k \in \{1,\ldots,N\} \setminus \{l | c_{il} = \perp\}} \frac{(x_k - c_{ik}^2)}{\sigma^2}). \tag{6}$$

A similar problem arises for MISO systems with an m dimensional output and rules which only effect l of these values. Again, with $\perp$ denoting an undefined value, we set o_{ik}, the k-th component of the i-th output vector to

$$o_{ik} = \begin{cases} f_{ik} & : \text{ k-th output component in consequent of the } i\text{-th rule} \\ \perp & : \text{ k-th output component not effected by } i\text{-th rule} \end{cases} . \tag{7}$$

The k-th output of the FRBF controller is then obtained as

$$y_k = \frac{\sum_{i \in \{1,\ldots,N\} \setminus \{l | o_{ik} = \perp\}} o_{ik} h_i(x)}{\sum_{i \in \{1,\ldots,N\} \setminus \{l | o_{ik} = \perp\}} h_i(x)}, \tag{8}$$

where the activation function $h_i(x)$ is calculated according to (6).

1.2.2 Exact Interpolation

Another problem with RBF networks used for Sugeno-type fuzzy control is that the networks do not exactly interpolate between the output vectors (consequent values of the rules) but rather perform an approximation. This is due to the

overlap of the basis functions at their centers: Even if one activation function (degree of fullfilment of an antecedent) takes the value one, the output of the controller deviates from the corresponding consequent value. However, exact interpolation sometimes is necessary if the control function must fulfill certain constraints. In obstacle avoidance, for instance, this could be a rule which sets the forward velocity to zero if the frontal distance reaches a certain value. This important rule must not be biased by other rules and probably will not be subjected to adaptation as well. Our solution to the exact interpolation problem is to introduce a set Π of *exact nodes* and to modify the activation functions $h_i(x)$ by multiplying the original activation functions with the complement of each activation function of nodes in Π, i.e.

$$h_i'(x) = h_i(x) \prod_{j \in \Pi \setminus i} (1 - h_j(x)), \tag{9}$$

which assures that if one membership function has degree of fulfillment one, the others will be suppressed and hence do not contribute to the mapping. For exact nodes we require that they have no undefined components in their output vectors and centers. A formal proof of the exact interpolation property can be found in [5].

The complete FRBF architecture is illustrated in Figure 1.

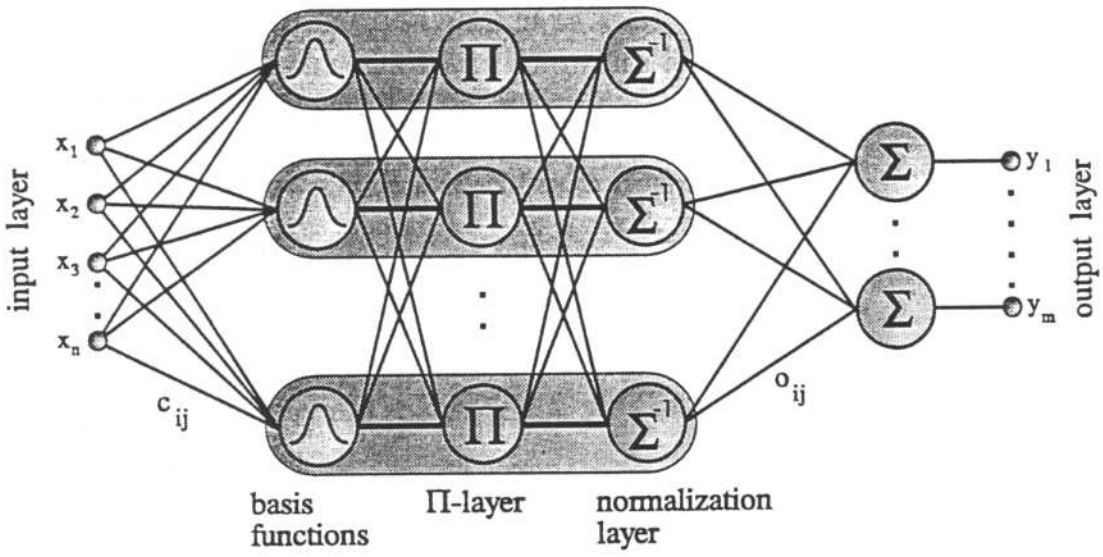


Figure 1: FRBF architecture with exact nodes: Each formal node in the nework consists of a basis function, a π neuron for exact interpolation and a normalization unit, which may be grouped in three computational layers.

1.2.3 Topology Preservation

DCS networks represent an extension of RBF-networks in that each node in the network gets to know (learns) which other nodes in the network are its nearest

neighbors in input space. The rationale for this is that activation of the nodes decreases with decreasing distance of the input from their centers, and hence only the best matching unit *bmu* (i.e. the node whose center c_i has smallest Euclidean distance to the input value x) and its direct neighbors significantly contribute to the mapping and need to be evaluated. But how can direct neighborhood be defined? The intuitive definition was provided by T. Martinetz, [6], calling two nodes neighbored in input space if their masked Voronoi cells[1] have a common border. This leads to the definition of the *induced Delaunay triangulation* or *perfectly topology preserving map* in which two nodes are connected if their masked Voronoi cells share a common border. If $G = (V, E)$, $V = \{1, \ldots, N\}$ is the graph representing the induced Delaunay triangulation of the centers $c_1, \ldots, c_N$ of the network, then the set of direct neighbors of node i, $Nh(i)$, can be defined as the direct neighbors w.r.t. G, i.e.

$$Nh(i) = \{j | (i, j) \in E\}. \tag{10}$$

The output of the network is calculated as

$$y(x) = \frac{\sum_{i \in Nh(bmu)} o_i h_i(x)}{\sum_{i \in Nh(bmu)} h_i(x)} \tag{11}$$

and includes only the best matching nodes (rules). This can lead to a significant speed up in output calculation.

The induced Delaunay triangulation (i.e. the graph G) for a given set of centers can be either calculated in advance or can be approximated with a simple learning rule that, on presentation of an input x, always connects the best and the second best matching unit, i.e. starting with $E = \emptyset$

$$E = E \cup \{(bmu, smu), (smu, bmu)\}. \tag{12}$$

If the input probability density function is different from zero for all possible inputs and the distribution of centers is *dense*[2], G converges to the induced Delaunay triangulation with probability one. If the centers (membership functions) are allowed to adapt, existing edges may need to be removed from G. This can be achieved by decaying existing edges by a learning rule such as proposed in [7].

Combining DCS networks with the ideas presented so far (incomplete rules and exact interpolation) we obtain the Fuzzy DCS (FDCS). Yet there remains a problem within the FDCS framework: How can a best matching unit be determined (and the topology be defined and learned) if components of the centers may be undefined? The intuitive answer is to replace the Euclidean metric with the distance measure

$$d(x, c_i) = \sqrt{\sum_{k \in \{1,\ldots,n\} \setminus \{l | c_{il} = \perp\}} (x_k - c_{ik}^2)}, \tag{13}$$

[1] If $M \subset \mathbb{R}^n$ is the input manifold and $V_i \subset \mathbb{R}^n$ the Voronoi cell of c_i then the maksed Voronoi cell is $V_i^{(M)} = V_i \cap M$.

[2] The distribution of centers is dense if for each possible input $x \in M$ the triangel formed by x, c_{bmu} and c_{smu} completely lies in M.

which takes into account only the defined components. With help of this distance measure, the notions of topology preservation and the learning rules for the topology preserving lateral connection structure as discussed in 1.2.3 can be generalized for the new situation of undefined components, as has been formally proven in [5].

2 Behavior-Based FRBF Control for Collision Avoidance

After this short introduction into our FRBF-based approach to fuzzy control we will now propose a fully reactive and behavior-based control architecture for obstacle avoidance.

The main problem with behavior-based control is the switching between different behaviors and that in order to exploit the advantages of smooth fuzzy control this switching additionally has to be smooth. We solve this problem by introducing a *pain* function which smoothly switches between the avoidance behavior and the task-driven behavior. The task-driven behavior consists of two subtasks, wall-following in an exploration phase or goal-following to reach a goal position.

2.1 FRBF-Based Architecture for Reactive Collision Avoidance

In the behavior-based subsumption-control-architecture, e. g. [8], the behaviors are organized horizontally, i.e. each behavior has full access to all sensor readings and proposes its own motor control command. The final motor control command is then computed by suppression: Low level behaviors are more important for the safety of the system and therefore have a higher priority than higher level tasks.

Classical subsumption architectures rest on a hard selection of one of their behaviors. Hence these architectures result in a kind of *bang-bang* control on the basis of potentially smoothly controlled *bang*-functions.

In fuzzy control we seek for a smoothly controlled decision between the authorization of two fuzzy controllers. To this end, let us consider two FRBF networks $\mathcal{A}$ and $\mathcal{B}$ which represent two hierarchically ordered behaviors. Here, $\mathcal{A}$ denotes the lower level behavior, more important for the safety of the system, and $\mathcal{B}$ denotes the more task oriented higher level behavior. We write $y_{\mathcal{A}} : I_{\mathcal{A}} \to O$ and $y_{\mathcal{B}} : I_{\mathcal{B}} \to O$ for the corresponding evaluation functions of both networks. The input spaces might even be different, so that the behaviors can be optimally adapted to their special tasks.

For each behavior we now introduce a *state-evaluation* function,

$$eval : I_{\mathcal{A}} \to [a, b], \; a, b \in I\!R, \tag{14}$$

which signals the danger of the current system state from the point of view of

each individual beahvior. The state-evaluation can be regarded as signalling the responsibility of behavior $\mathcal{A}$.

In biological systems we also have a kind of state-evaluation which can be regarded as a pain signal. Pain is able to change the behavior of any living being, if only the pain intensity exceeds some threshold value. Accordingly, we introduce a *pain signal*, $p : [a, b] \rightarrow [0, 1]$, whose response activates the behavior.

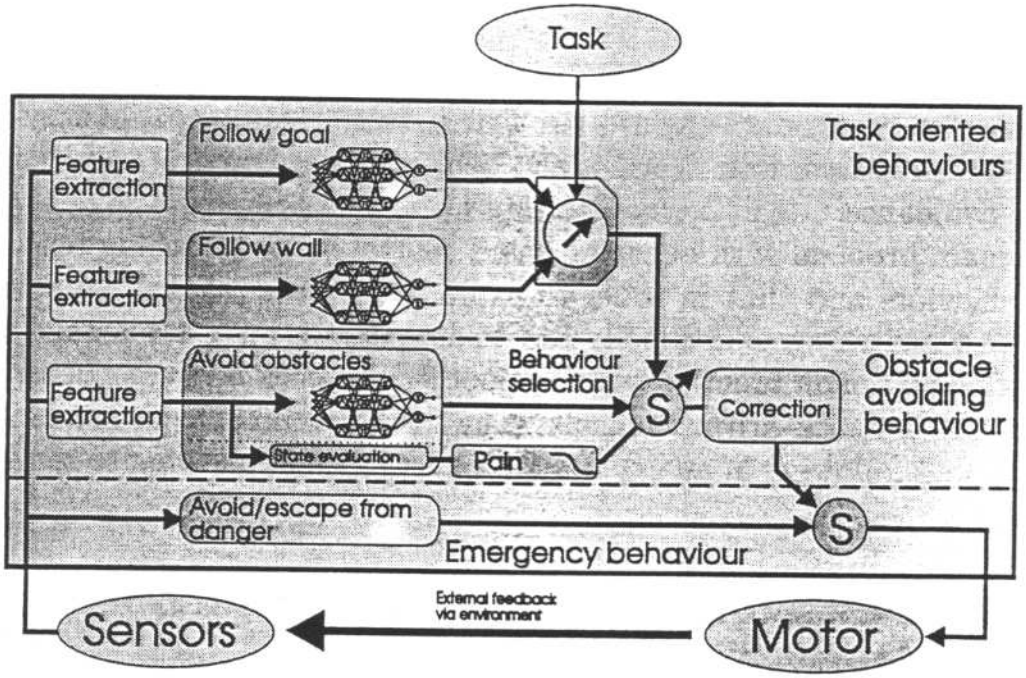


Figure 2: Behavior-based fuzzy architecture for obstacle avoidance

We demand p to be a monotonically decreasing function fulfilling $p(a) = 1$ and $p(b) = 0$. Using the convex-combination

$$y = p(eval(x_{\mathcal{A}}) \cdot y_{\mathcal{A}}(x_{\mathcal{A}}) + (1 - p(eval(x_{\mathcal{A}})) \cdot y_{\mathcal{B}}(x_{\mathcal{B}}), \tag{15}$$

we obtain a motor control command $y \in O$ for any input $x_{\mathcal{A}} \in I_{\mathcal{A}}$ and $x_{\mathcal{B}} \in I_{\mathcal{B}}$. For a low state evaluation $eval(x_{\mathcal{A}})$ the behavior $\mathcal{A}$ dominates, while for high state evaluations the resulting reaction will be dominated by the higher level control network. The values a and b represent the maximum and minimum state evaluation. As an example of a pain signal, consider

$$p(z) = \varphi(z) - \varphi(b) + \frac{\varphi(a) - \varphi(b) - 1}{b - a} \cdot (z - b), \text{ with} \tag{16}$$

$$\varphi(x) = \frac{1}{1 + \exp(\varrho \cdot (x - m))}. \tag{17}$$

The parameter $m \in [a, b]$ denotes the location of the reversal point, and $\rho \in \mathbb{R}$ a parameter that stands for the inclination of the pain signal. For $\rho = 0$, we obtain a simple linear function, and for $\rho \rightarrow \infty$ we get a threshold function with $p(z) = 1$ for $z < m$ and $p(z) = 0$ for $z > m$. Figure 3 illustrates the pain signal.

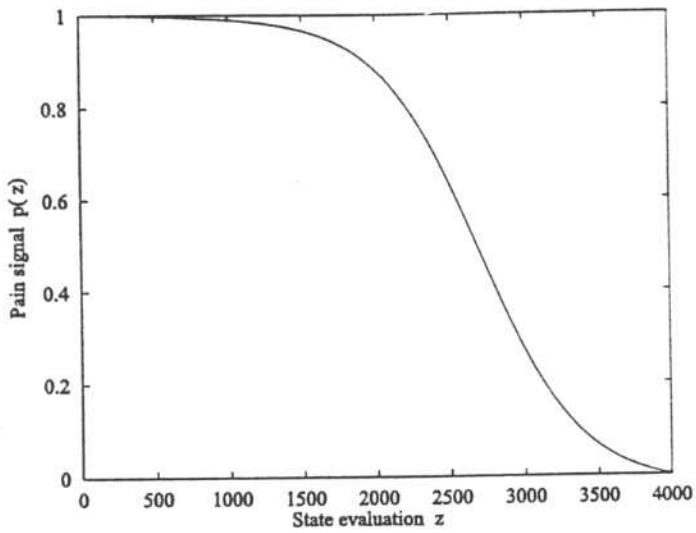


Figure 3: Pain signal for the parameter set $a = 0$, $b = 4000$ and $\varrho = 0.003$

Using a pain signal, we can smoothly combine two different behaviors. Since these behaviors themselve consist of smooth FRBF controllers, we finally obtain a smoothly controlled behavior-based reactive control architecture.

For reasons of safety, however, we added a conventional hard-wired lowest-level emergency behavior that is able to avoid a direct imminent collision by applying an emergency stop and triggering an orientation behavior. Yet all higher level tasks, such as obstacle avoidance, wall following and goal following, result from a smooth combination of different fuzzy controllers.

2.1.1 Feature Extraction

Each FRBF-controller has its own pre-processing stage for feature extraction. During this stage from all available sensory inputs only those features are selected that are relevant for the specific task of each behavior. Such a pre-processing stage is necessary to reduce the dimensionality of the input spaces of the FRBF-controllers. In our experiments, the sensory input consists of all eight sonar readings, the actual translational velocity and the angular velocity (jog-rate). Not all these inputs are necessary for all behaviors.

For the FRBF-controller of the obstacle avoidance behavior we choose as features all eight sonar readings, whereas the features of wall following are just the orientation to the left and to the right wall.

Using the ultrasonic sensor arrangement as shown in Figure 4, we define these two orientation features by

$$\alpha_{right} = \arctan\left(\frac{sensor[5] - sensor[4]}{d_v}\right) \cdot \frac{180^\circ}{\pi}, \tag{18}$$

$$\alpha_{left} = \arctan\left(\frac{sensor[7] - sensor[6]}{d_v}\right) \cdot \frac{180^\circ}{\pi}, \tag{19}$$

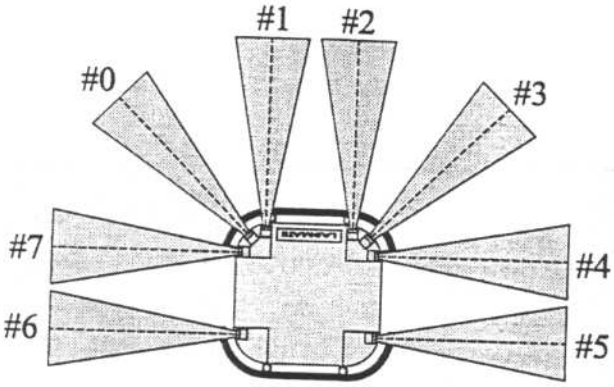


Figure 4: Arrangement of ultrasonic sensors for the TRC Labmate

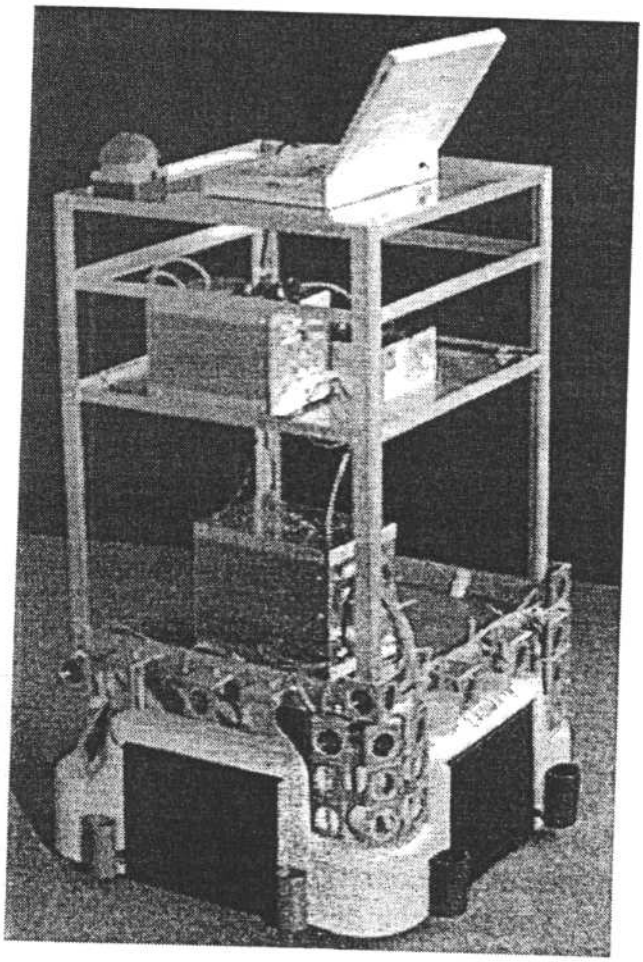


Figure 5: The physical robot

where d_v defines the vertical distance of the lateral sensors.

Finally, the feature extraction for the goal following behavior is canonically defined by the horizontal and vertical components Δx and Δy of the Euclidean distance between the center of the robot and the goal position in egocentric robot coordinates.

In addition to the features proposed here, additional features can be utilized. For example, similar to the inclination to lateral obstacles, the inclination to frontal obstacles can be defined as a possible feature. If we want to formulate fuzzy rules in which two sonar readings are compared (e.g. $s_1 > s_2$), we introduce the difference $(s_1 - s_2)$ as an additional feature.

2.1.2 Correction of Direct Motor Commands

An additional safety component checks whether the motor commands are within a tolerable interval or not. In particular, such post-processing is recommanded if the FRBF controller is allowed to adapt, cf. Section 4. In order to make sure that only reasonable outputs reach the motors, we filter the proposed commands by

$$\tilde{a} = \begin{cases} a & : \quad a \in [l_a, r_a] \\ l_a & : \quad a < l_a \\ r_a & : \quad a > r_a \end{cases} . \tag{20}$$

Filtering the proposed value $a \in \{\omega, v\}$ finally yields a corrected value $\tilde{a}$ which lies in the admissable intervall $[l_a, r_a]$. The value v denotes the velocity and ω the superimposed angular velocity.

A simple correction mechanism that guarantees a driving velocity with which the robot cannot contact an obstacle in the distance d_{stop} before T seconds is obtained by choosing

$$r_v = \frac{d_{min} - d_{stop}}{T}, \tag{21}$$

where d_{min} is the minimum frontal distance to any obstacle.

2.2 Fuzzy Rules for the FRBF Controller

In this section we provide the fuzzy rule base for the FRBF controller. Instead of linguistic terms we directly give numerical values. The values of the premises make up the centers, as described in Section 1.1. The sign $\approx$ will be used whenever conventional (inexact) neural units are used. Correspondingly, the use of exact neural units is denoted by the $=$ sign. Sonar readings are denoted by s_i.

2.2.1 Collision Avoidance Behaviors

Obstacle avoidance is often divided into sub-behaviors that treat special types of obstacle situations. For instance, in [8] one distinguishes between sub-tasks as *avoiding frontal obstacles* or *avoiding lateral obstacles.*

Our fuzzy controller has no inherent structure or hierarchy of rules, yet we arrange the rules in different groups of neural units that can be interpreted as *velocity control units*, *frontal collision avoiding units* and *lateral collision avoiding units.* The fuzzy rules are shown in Tables 1, 2 and 3. Figure 6 illustrates the evaluation function of the resulting FRBF controller.

if $(s_1 \approx 250)$ **then** $v \approx 0$
if $(s_2 \approx 250)$ **then** $v \approx 0$
if $(s_1 \approx 2000) \wedge (s_2 \approx 2000) \wedge (s_0 \approx 500) \wedge (s_3 \approx 500)$ **then** $v \approx 500$
if $(s_1 \approx 2000) \wedge (s_2 \approx 2000) \wedge (s_0 \approx 1000) \wedge (s_3 \approx 1000)$ **then** $v \approx 600$
if $(s_1 \approx 2000) \wedge (s_2 \approx 2000) \wedge (s_0 \approx 2000) \wedge (s_3 \approx 2000)$ **then** $(v = 1000) \wedge (\omega = 0)$
if $(s_1 \approx 1500) \wedge (s_2 \approx 1500)$ **then** $v \approx 400$
if $(s_1 \approx 1000) \wedge (s_2 \approx 1000)$ **then** $v \approx 300$
if $(s_1 \approx 500) \wedge (s_2 \approx 500)$ **then** $v \approx 150$

Table 1: Rules for controlling driving velocity

ω	s_0				
s_3	0	300	700	1200	1800
0	0	30	30	30	30
300	-30	0	30	30	30
700	-30	-30	0	30	30
1200	-30	-30	-30	0	30
1800	-30	-30	-30	-30	0
ω	s_1				
s_2	0	300	700	1200	1800
0	0	30	30	30	30
300	-30	0	30	30	30
700	-30	-30	0	30	30
1200	-30	-30	-30	0	30
1800	-30	-30	-30	-30	0

Table 2: Rules for avoiding frontal obstacles

ω	s_4			
s_5	0	300	700	1200
0	30			
300	30	30		
700	30	30	30	
1200	30	30	30	30
ω	s_7			
s_6	0	300	700	1200
0	-30			
300	-30	-30		
700	-30	-30	-30	
1200	-30	-30	-30	-30

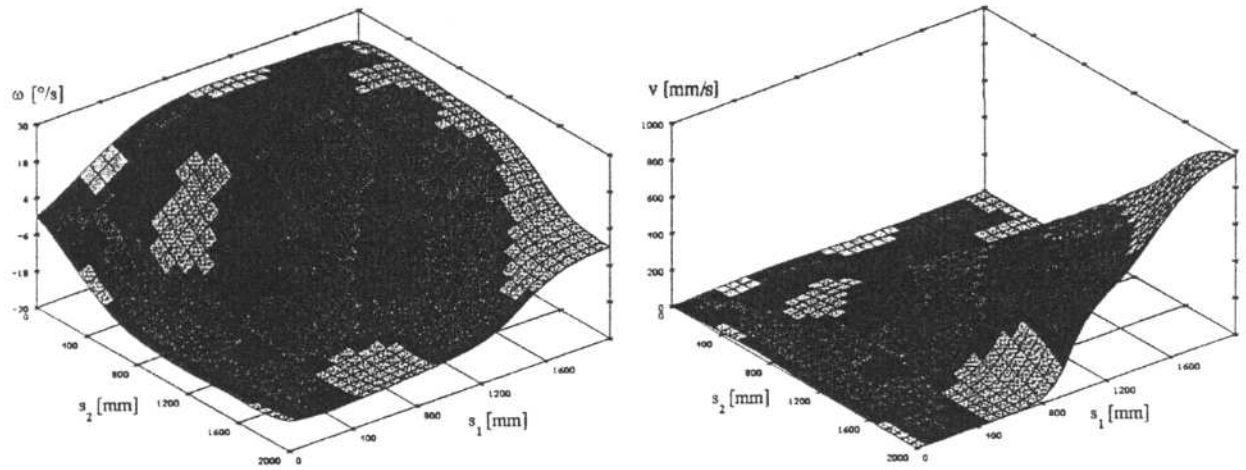
Table 3: Rules for avoiding lateral obstacles

2.2.2 Task Oriented Behaviors

Table 4 shows the rule base for the wall following and the goal-following behavior. These rules are realized by exact neural units. Only by using exact neural units we can assure that the driving velocity of the vehicle exactly becomes zero at the goal position. Due to superposition of several units this would hardly be possible with inexact neural units.

2.3 Simulations

In our simulations, the FRBF-based approach for obstacle avoidance turned out to be superior to the classic behavior-based approaches, e. g. [8], we investigated for comparison. The FRBF-based approach was able to achieve a higher maximum speed when there were only far obstacles. In contrast to the maximum speed of $0.25\frac{m}{s}$ achieved by the classic approach, the fuzzy controller



a) b)

Figure 6: Evaluation function of the FRBF controller for fixed input values $s_0 = s_3 = \cdots = s_7 = 2000\ mm$.

Goal following			
v/ω	Δx		
Δy	-2000	0	2000
-2000	0/-30	0/30	0/30
0	0/-30	0/0	0/30
2000	400/-30	800/0	400/30

Wall following			
v/ω	α_{left}		
α_{right}	-50	0	50
-50	800/-30	800/-30	
0	800/-30	800/0	800/30
50		800/30	800/30

Table 4: Fuzzy rules for wall- and goal following behavior

nearly reached the maximum speed of $1\frac{m}{s}$. Nevertheless it was careful in narrow environments where it navigated with very low speed. In extremely narrow environments, the FRBF network controls the forward velocity down to zero and a remaining rotational component results in an "escape from danger" behavior.

Furthermore the smooth control by the FRBF controller was observed to produce fewer oscillations during a corridor passage. The wall following behavior is able to align the vehicle with walls. Figure 7 shows the robot in a simulated environment. The velocity can be recognized from the distance of the dashes perpendicular to the driving direction. Higher velocities produce more distant dashes while lower velocities produce narrower dashes. In environments with far obstacles the vehicle reaches a velocity of nearly $0.8\frac{m}{s}$.

For all FRBF controller we used Gaussian neural activation functions with a standard deviation of $\sigma = 0.2$. Inputs to the network were normalized. As a state evaluation we defined

$$eval(s_0, \cdots, s_7) = \sqrt{s_0^2 + s_1^2 + s_2^2 + s_3^2}. \tag{22}$$

The maximum measurable distance (timeout distance) of the ultrasonic sensors was set to $2m$, so the parameter a and b were set to $a = 0$ and $b = 4000$. For the pain signal we have chosen $m = 2700$ and $\varrho = 0.003$. The time step width was

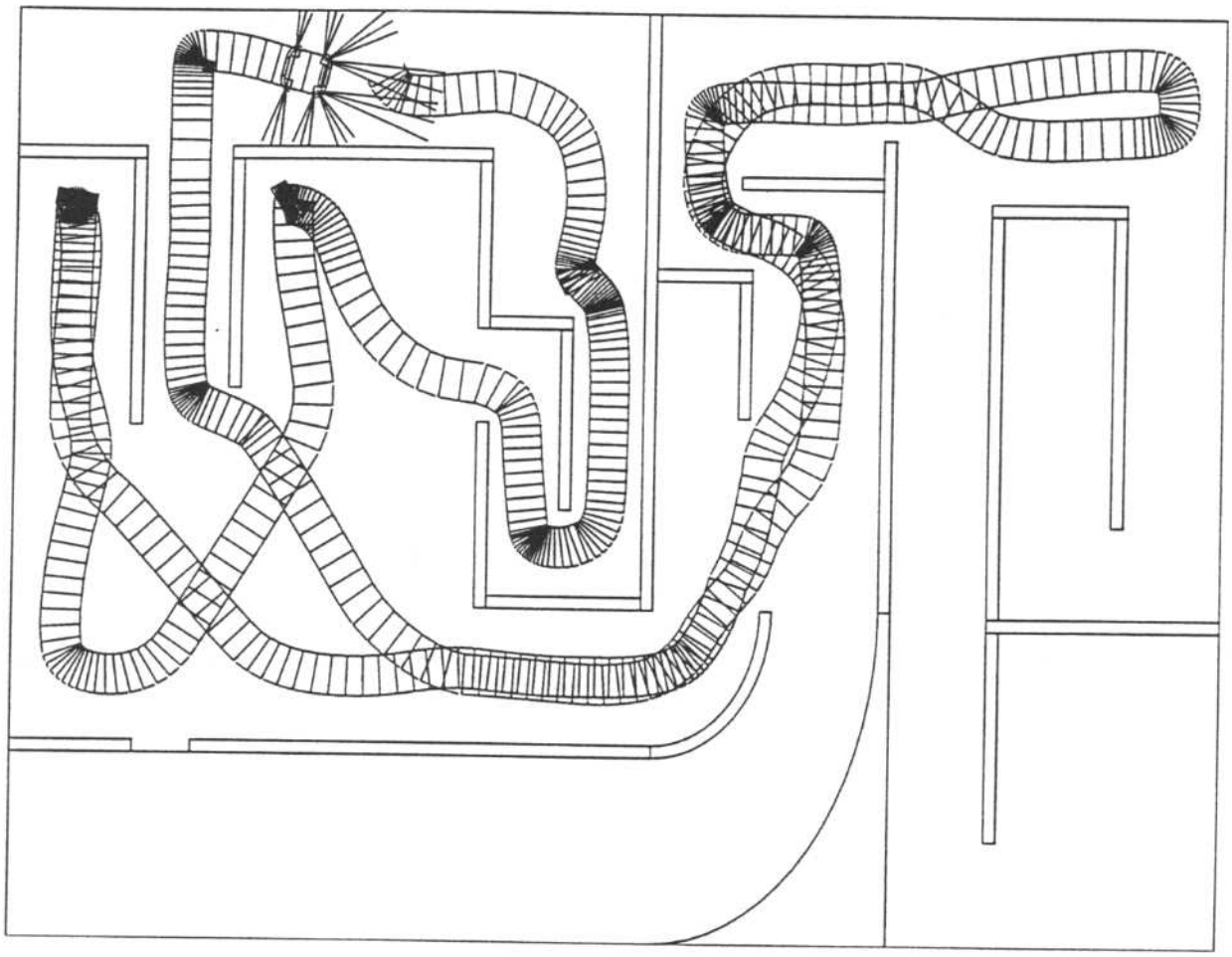


Figure 7: FRBF controlled tour of a simulated autonomous mobile robot

set to $0.4s$ and corresponds to the time interval for one control cycle in reality.

There is, however, a problem with the proposed control scheme which has to be addressed. In contrast to the classical subsumption architecture where each behavior proposes its own motor command and finally only one command is selected by a hierarchy of suppression there is no possibility of prefering specific rules in the FRBF network architecture. All neural units contribute their activation-driven output to the motor command, which is finally send to the motor. For example, in the classical subsumption architecture testing whether left rotation is an appropriate reaction before testing for right rotation results in a behavior which tends to prefers left curves . Such behavior can be regarded as *non-symmetric*. On the contrary, the FRBF architecture has no such built in preference and will typically result in a *symmetric* behavior (cf. Figure 6).

Symmetric controllers cause an undesirable behavior (in simulations as well as in reality) when the robot is driving into a corner. For simplicity we only consider the sonar readings # 1 and # 2. At first, both readings decrease and as a result the vehicle decreases its velocity. If the left sensor shows a smaller reading than the right sensor, the vehicle will turn to the right and vice versa. But now, by turning towards one direction, because of the new environmental

situation the state of the sonar readings will switch and the symmetric controller will turn to just the opposite direction in the next step. This behavior results in a kind of *dead lock situation*, shown in Figure 8, which normally should be avoided. Until now, only the hard wired emergency behavior can free the robot from the dead lock situation.

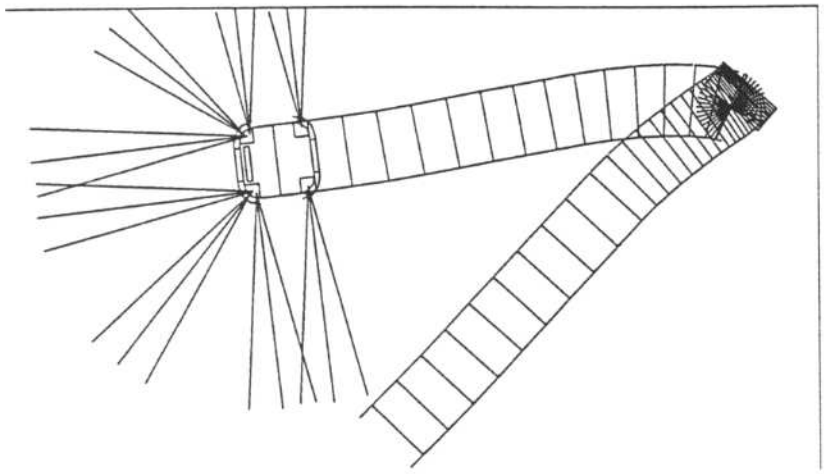


Figure 8: A dead lock situation

3 Freespace-Based FRBF Control for Collision Avoidance

To overcome the dead lock problem mentioned above and keep on profiting from the advantages of fuzzy control, we next propose a FRBF based approach that uses improved feature extraction from sonar readings and only consists of a single FRBF network.

3.1 Feature Extraction

In order to avoid a symmetric fuzzy controller we change the input features of the controller and introduce an angle that points into the direction of a free space that is closest to the heading of the vehicle. This free space direction provides an additional input feature and can be obtained by a technique which is similar to the obstacle avoidance approach of [9].

In [9], the last three sonar readings are stored and build a coarse temporally and spatially restricted model of the environment. From this model an optimal trajectory, chosen from a set of circles through the center of the robot and tangential to the heading direction, is generated and provides the obstacle avoiding behavior.

What we did is to extract the free space direction from the last stored sensor readings and provide it as an additional input feature for our fuzzy controller.

Because no reliable free space information can be estimated from only eight ultrasonic sensor readings, we also store the sonar readings in a short time memory. From this short time memory we extract a coarse model of the direct robot surrounding. In contrast to [9] we do not use this model to explicitly estimate a trajectory, but we recover the free space direction closest to the actual heading direction. Figure 9 shows the short time representation of the environment from which a free space direction is generated.

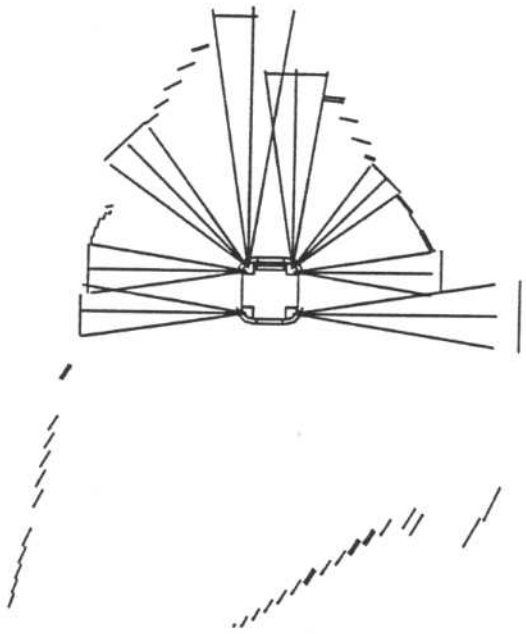


Figure 9: Short time representation of the robot surrounding

From the short time memory, additional features can be extracted, e. g.

- Direction of the free space closest to the goal direction. This will result in a goal following behavior.
- Direction of the free space closest to the actual robot heading. This will result in a wandering behavior.
- Distance to frontal or lateral obstacles.
- Orientation of frontal or lateral walls.

3.2 FRBF-Based Architecture for Collision Avoidance with Freespace Information

In this section we investigate FRBF control based on free space extraction for obstacle avoidance and show that it can solve the dead lock problem of the behavior-based approach.

Eight sonar readings and the supplementary extracted free space direction make up the new input features. As in the previous section two higher level

tasks are considered, namely wall alignment (in order to provide an exploration behavior) and goal following. So far these two tasks have been divided into two different control modules. Using the additional free space information, this division is no longer necessary. Both tasks can be solved by choosing the free space direction in accordance with the particular task.

Goal following behavior: If the robot has to reach a specified position, the free space direction is selected that is closest to the direction of the goal position.

Wandering around behavior: Exploration of the robot surrounding can be achieved by a free space direction that is closest to the robot heading. If the robot moves towards a wall, the free space direction will align to the wall direction. Therefore, following the free space direction produces a wall following behavior.

The resulting controller architecture consists of a single FRBF controller together with the correction stage and the additional hard-wired emergency behavior on the lowest level. The new architecture is shown in Figure 10.

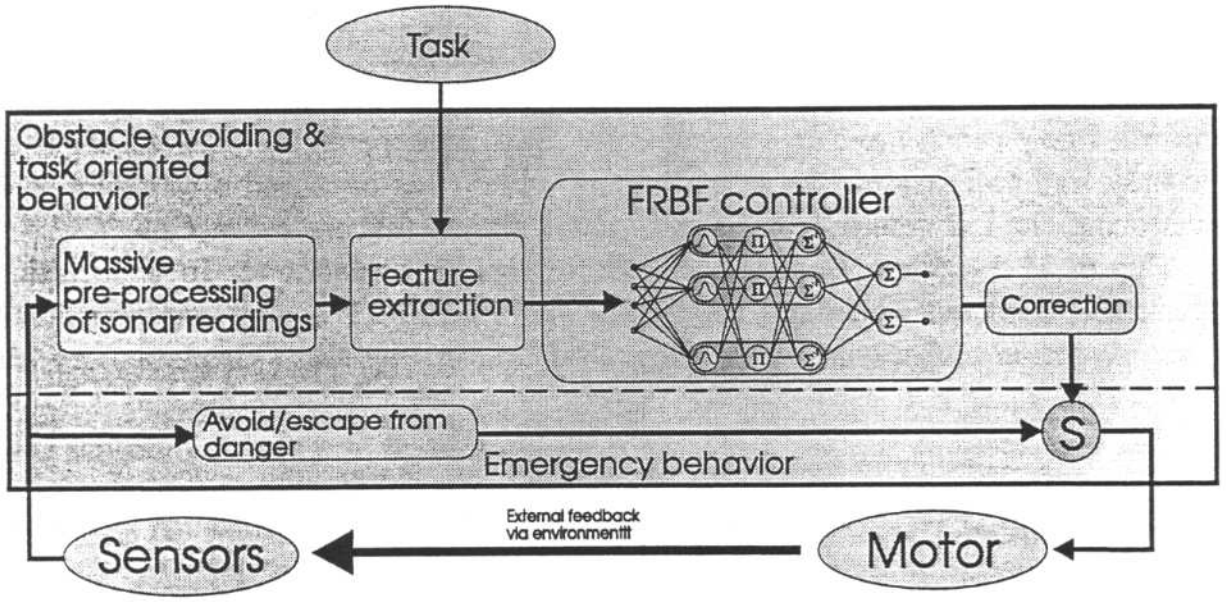


Figure 10: FRBF-based obstacle avoidance using massive sonar reading pre-processing

3.3 Fuzzy Rules of the FRBF Controller

Tables 5-8 declare the fuzzy rules of the FRBF controller used for free space driven obstacle avoidance. As input features we use all eight sonar readings $s_0, \cdots, s_7$, as well as the free space direction α_{best} according to the specified task. Supplementary, we use the rules of Table 3. The fuzzy rules of Table 8 represent exact neural units.

v/ω	α_{best}					
	-150	-100	-50	50	100	150
$s_1 \approx 100$:	0/-30	0/-30	0/-30	0/30	0/30	0/30
$s_2 \approx 100$:	0/-30	0/-30	0/-30	0/30	0/30	0/30

Table 5: Rules dealing with emergency situations

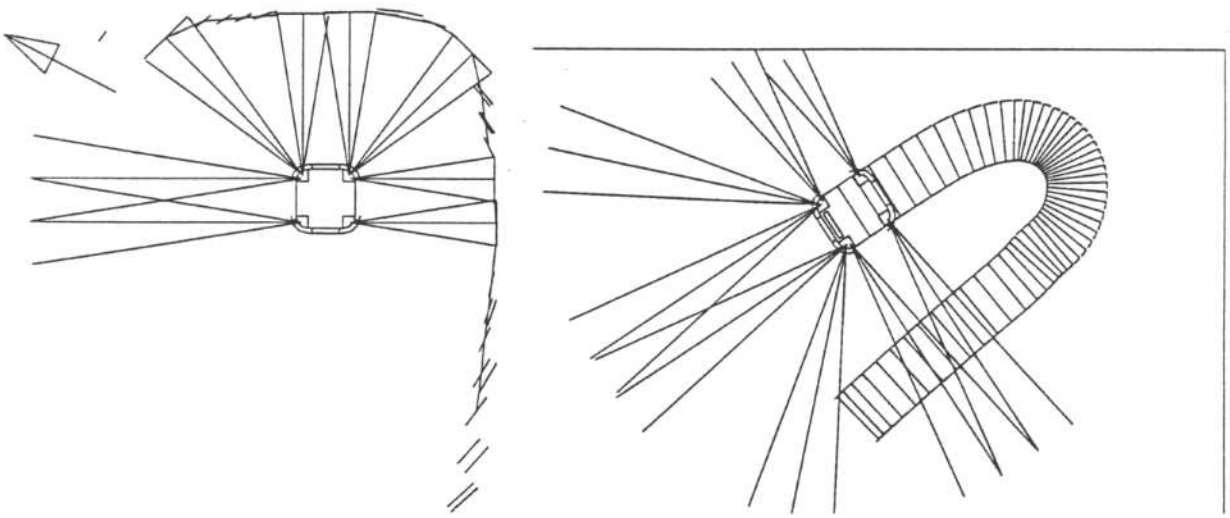
if $(s_1 \approx 3000) \wedge (s_2 \approx 2000) \wedge (s_0 \approx 600) \wedge (s_3 \approx 600)$ **then** $v \approx 600$
if $(s_1 \approx 3000) \wedge (s_2 \approx 2000) \wedge (s_0 \approx 1300) \wedge (s_3 \approx 1300)$ **then** $v \approx 750$
if $(s_1 \approx 3000) \wedge (s_2 \approx 3000) \wedge (s_0 \approx 3000) \wedge (s_3 \approx 3000)$ **then** $(v = 1000) \wedge (\omega = 0)$
if $(s_1 \approx 1500) \wedge (s_2 \approx 1500)$ **then** $v \approx 400$
if $(s_1 \approx 1000) \wedge (s_2 \approx 1000)$ **then** $v \approx 300$
if $(s_1 \approx 500) \wedge (s_2 \approx 500)$ **then** $v \approx 150$

Table 6: Rules for controlling the driving velocity

3.4 Simulations

The experiments we have carried out applying the above FRBF-based architecture demonstrate the workability of our approach, i.e. to combine the benefits of smooth fuzzy control and to overcome the dead lock problem. Remember that we only had to introduce an extra feature which was obtained straightforward by storing the last sensor readings and some simple feature extraction.

Figure 11 b) demonstrates a successful avoiding maneuver in a situation where the previous approach has failed. In Figure 11 a) the extracted free space direction and the sensor readings as stored in the short time memory are depicted.



a) b)

Figure 11: a) Extracted free space direction b) Simulated avoiding maneuver

$\alpha_{best} \approx 0$:					
ω	s_0				
s_3	0	300	700	1200	1800
0		30	30	30	30
300	-30		30	30	30
700	-30	-30		30	30
1200	-30	-30	-30		30
1800	-30	-30	-30	-30	

$\alpha_{best} \approx 0$:					
ω	s_1				
s_2	0	300	700	1200	1800
0		30	30	30	30
300	-30		30	30	30
700	-30	-30		30	30
1200	-30	-30	-30		30
1800	-30	-30	-30	-30	

ω	α_{best}					
$s_0 = s_3$	-150	-100	-50	50	100	150
0	-30	-30	-30	30	30	30
300	-30	-30	-30	30	30	30
700	-30	-30	-30	30	30	30
1200	-30	-30	-30	30	30	30
1800	-30	-30	-30	30	30	30

ω	α_{best}					
$s_1 = s_2$	-150	-100	-50	50	100	150
0	-30	-30	-30	30	30	30
300	-30	-30	-30	30	30	30
700	-30	-30	-30	30	30	30
1200	-30	-30	-30	30	30	30
1800	-30	-30	-30	30	30	30

Table 7: Rules for avoiding frontal collisions

4 Reinforcement Learning of an FDCS Controller for Collision Avoidance

The reason for using *neuro*-fuzzy control is to allow for adaptation. In this section we want to briefly[3] demonstrate the learning capabilities of an FDCS controller by reinforcement learning of collision avoidance. Circumstances are very hard because i) the reinforcement signal provides only minimal feedback from the environment, ii) the reinforcement signal is delayed, iii) we start with only one fuzzy rule and the learner has to generate new rules from interaction

[3] A more detailed description of this experiment as well as learning parameters can be found in [10] and [5] .

ω	α_{best}					
	-150	-100	-50	50	100	150
$s_1 \approx s_2 \approx 3000$:	-30	-30	-30	30	30	30

Table 8: Rules for free space following

with the environment, iv) we allow for adaptation of membership and consequent functions and v) we additionally learn the topology of the input space and hence an FDCS controller, cf. Section 1.2.3, for accelerated output calculation. On the one hand, our experiments confirm that even in this extreme setup the controller is indeed able to learn an obstacle avoidance behavior. On the other hand, our experiments also underline that pure reinforcement learning (without prior knowledge and without additional learning mechanisms) takes too much time and too many trial to be of much practical value.

4.1 Learning Architecture and Algorithms

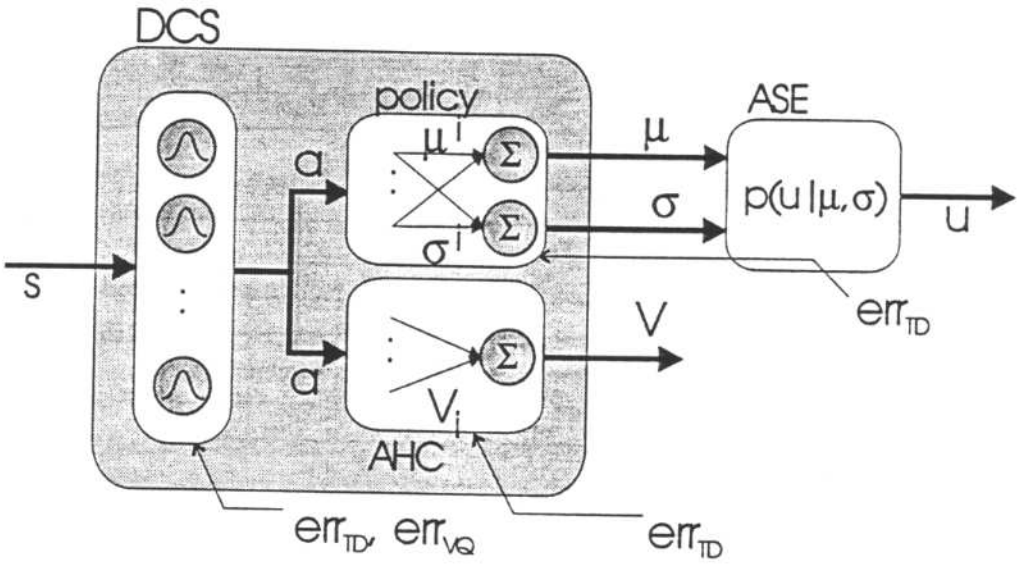


Figure 12: FDCS-based reinforcement learning architecture

As depicted in Figure 12 the controller is realized by a single DCS network with an additional stochastic associative search element (ASE) for REINFORCEment learning. The DCS network implements both the actual controller (policy) as well as an adaptive heuristic critique (AHC), [11]. The calculation of the control vector u given the sensory input s proceeds as follows:

First, the input vector is transformed to an activity vector representing the normalized activations (degrees of fulfillment) of the RBF units (rules) with centers c_i and uniform width:

$$a_i = \frac{h_i(s)}{\sum_{j \in Nh(bmu)} h_j(s)} \,. \tag{23}$$

Second, a *prototypical action vector* μ, a *certainty vector* σ and the *predicted cumulative reinforcement* V are calculated by a weighted sum of contributing

vectors (consequent functions) attached to the RBF units[4]:

$$\mu = \sum_{i \in Nh(bmu) \cup \{bmu\}} a_i \mu^i \ , \ \sigma = \sum_{i \in Nh(bmu) \cup \{bmu\}} a_i \, \sigma^i \ , \ V = \sum_{i \in Nh(bmu) \cup \{bmu\}} a_i \, V_i. \tag{24}$$

Finally the ASE draws the actual action vector u according to a Gaussian probability density function with marginal distributions

$$p(u_j|s) = p(u_j|\mu_j(s), \sigma_j(s)) = \frac{1}{\sqrt{(2\pi)}\sigma_j} e^{-\frac{(u_j - \mu_j)^2}{2\sigma_j^2}}. \tag{25}$$

On-line adaptation is performed w.r.t. the contributing prototypical action vectors μ^i, certainty vectors σ^i and evaluation values V_i attached to the RBF units (consequent - part) as well as to the centers c_i of the RBF units (antecedent - part). The evaluation values V_i are updated using a TD(1) rule [12]:

$$\Delta V_i = \alpha_V \, err_{TD}(s_t) \sum_{k=1}^{t} (\lambda\gamma)^{t-k} \nabla_{V_i} V(s_k) \tag{26}$$

$$= \alpha_V \, (\gamma V(s_{t+1}) + r_t - V(s_t)) \sum_{k=1}^{t} (\lambda\gamma)^{t-k} a_i(s_k). \tag{27}$$

where $err_{TD}(s_t) = r_t - b_t$ denotes the current temporal difference error with r_t the reinforcement signal and $b_t = V(s_{t+1}) - V(s_t)$ an adaptive baseline. Prototypical action vectors μ^i and certainty vectors σ^i are adapted using a REINFORCE gradient descent:

$$\Delta \mu_j^i = \alpha_\mu \, err_{TD}(s_t) \frac{\partial \ln p_j(u_j|\mu_j, \sigma_j)}{\partial \mu_j} \frac{\partial \mu_j}{\partial \mu_j^i} \tag{28}$$

$$= \alpha_\mu \, (\gamma V(s_{t+1}) + r_t - V(s_t)) \frac{(u_j - \mu_j)}{\sigma_j^2} a_i \tag{29}$$

and

$$\Delta \sigma_j^i = \alpha_\sigma \, err_{TD}(s_t) \frac{\partial \ln p_j(u_j|\mu_j, \sigma_j)}{\partial \sigma_j} \frac{\partial \sigma_j}{\partial \sigma_j^i} \tag{30}$$

$$= \alpha_\sigma \, (\gamma V(s_{t+1}) + r_t - V(s_t)) \frac{(u_j - \mu_j)^2 - \sigma_j^2}{\sigma_j^3} \, a_i \tag{31}$$

The REINFORCE framework [13] states that (28) and (30) implement a gradient descent on the expected reinforcement (at least for a constant baseline b). When the algorithm converges towards a local maximum of the reinforcement the σ_j^i will decrease to small values, narrowing the range of stochastic search. Hence the

[4] the contributing prototypical action vectors and certainty vectors are denoted by superscripts.

term certainty values: If we pre-structure the network with fuzzy rules we can specify the search range for the conclusion of this rule by specifying its σ^i vector. Values close to zero result in non-changing consequents (fixed rules). On the other hand, if we analyze the network at consecutive time steps, non decreasing components of σ^i_j indicate convergence to (certainty about) the corresponding prototypical action.

Finally, the centers c_i of the *bmu* and its topological neighbors are updated according to an error modulated Kohonen rule as described in [7]:

$$\Delta c_i = \epsilon_{modulated}\, err_{VQ} \quad \text{with} \quad err_{VQ} = s - c_i \tag{32}$$

The error we use for modulation is the TD-error which is locally accumulated for every node in the DCS network. The lateral connection structure of the DCS is adapted with a learning rule derived from (12), again refer to [7] for details. A new neural unit (rule) is inserted whenever the distance to the current best matching unit is too large. At most N_{max} units are inserted.

4.2 Experiments

In order to test the applicability of our learning controller to collision avoidance with the TRC Labmate the simulated Labmate was placed in the training environment depicted in Figure 2. The Labmate was then allowed to drive around until either the distance to an obstacle dropped below $20cm$ or 200 time steps elapsed, ending a trial. In the former case, an orientation behavior is triggered which causes the Labmate to rotate until the front sensors indicate free space. In the latter case the Labmate is stopped and rotated for a random angle (to prevent it from staying on a closed trajectory all the time). Since we want to test the performance of the controller independent of incorporated prior knowledge the controller started with only one fuzzy rule:

if $(s_0 \approx 5000) \wedge \cdots \wedge (s_7 \approx 5000)$ **then**

$(\mu_v(s) \approx 400) \wedge (\sigma_v(s) \approx \sigma_v^0) \wedge (\mu_\omega(s) \approx 0) \wedge (\sigma_\omega(s) \approx \sigma_\omega^0) \wedge (V(s) \approx 0)$,

stating that if all sensor readings are about $5m$ the Labmate should drive forward (zero angular velocity μ_ω) with velocity $\mu_v = 400\ cm\, s^{-1}$. The certainty values for forward velocity and angular velocity $(\sigma_v, \sigma_\omega)$ were set to small initial values $(\sigma_v^0, \sigma_\omega^0)$. As immediate reinforcement we used the difference between evaluations of two succeeding situations, $r_t = \Phi(s_{t+1}) - \Phi(s_t)$, with $\Phi : \mathbb{R}^8 \rightarrow \mathbb{R}$ the evaluation function of a sensory situation. In addition the Labmate was given a high negative reinforcement signal if it had approached an obstacle within less than $20cm$.

For a typical run, Figure 13 shows the Labmate at the beginning of training in the training environment. Figure 14 shows collision free navigation of the Labmate in a test environment after training phase. End of training is indicated by the averaged TD-error approaching a minimum, the averaged reinforcement approaching its maximum and - of course - avoidance of collisions. Plots for the TD-error and the reinforcement (both averaged over 100 trials) are depicted in Figure 15. In our experiments training took between 1000 and 10000 trials, taking (on average) a longer time when sonar sensors with a characteristic beam

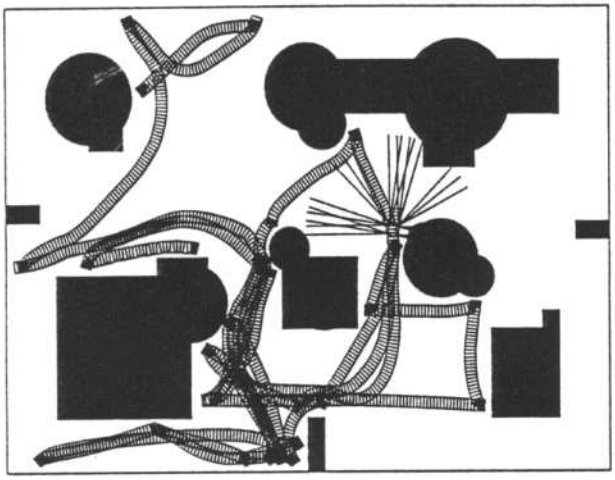


Figure 13: start of training, training environment

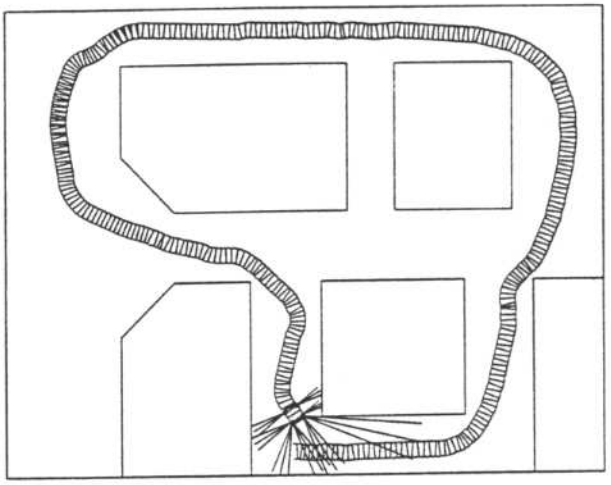


Figure 14: end of training, test environment

width of 20° and 5% noise were simulated than simulating idealized sensors (0° beam width) without noise. However, the difference between these two types of simulated sensors turned out to be surprisingly small. At most $N_{max} = 100$ neural units (rules) have been utilized. No effort has been spent on parameter optimization.

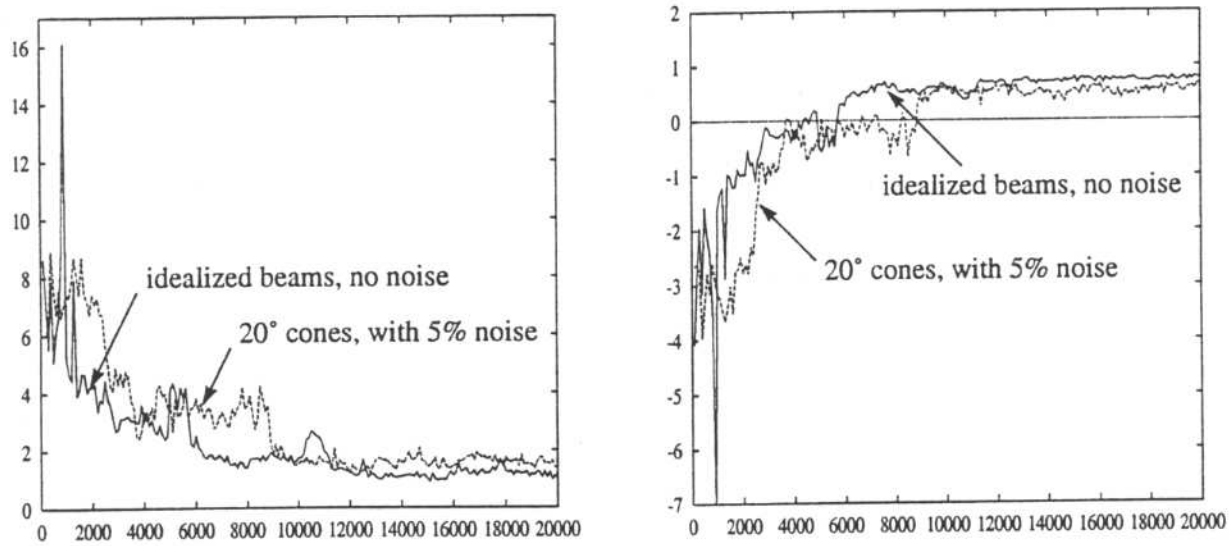


Figure 15: TD-error (right) and reinforcement (left) versus number of trials

5 Sensor Preprocessing

In most of the work on fuzzy logic control of mobile robots, sensor pre-processing is either overlooked or highly simplified, assuming that the sensors deliver exact (true) values [14, 15, 16, 17, 18]. On the contrary, we argue that it deserves equal attention as the controller whose performance crucially depends on the accuracy of the sensors.

There are two main reasons that necessiate sensor pre-processing in sonar-based fuzzy logic control. First, due to crosstalk, bad reflection properties of the environment and shielding problems, sonar sensors frequently produce highly erroneous readings which must be filtered out prior to using them for reactive control. Second, when the dimension of the input space becomes too high, it does no longer make sense to treat each input as a separate variable in fuzzy control. Besides the exponential growth of the number of rules it will be difficult to attribute a meaning to every input (which, however, is crucial for fuzzy control). In case of more than eight sonar sensors the solution to the second problem is a meaningful grouping of the sensor readings, which may be regarded as a kind of abstraction from the sensor readings using prior knowledge.

Such abstraction is not unique to the fuzzy control but has been addressed in connectionist inductive learning too. For instance, multi-layer networks have been trained in mobile robot navigation tasks, where the hidden layers construct a generalized intermediate representation of the input by supervised learning, [19, 20]. Likewise, reinforcement learning techniques have been employed to generalize the input space by recursively partitioning the state space based on the individual bit relevance [21]. However, a common characteristic of the above generalization techniques is that input uncertainties are not considered and domain knowledge is largly ignored.

The work in this section proposes a sensor pre-processing method for our TRC mobile robot. It combines *domain knowledge* and Kalman filtering to condense the sensory data and to cope with the uncertainties of the readings. The proposed pre-processing has been successfully tested on the real robot using an FRBF controller.

5.1 Partitioning the Perceptual Space

Instead of eight sensors in the previous sections (see Figure 4) in this section we utilize 10 sonar sensors. The sensors cover a total frontal angle of 120 *degrees* and are pre-programmed to measure a distance up to $2m$. Even assuming ideal sensors and the simplest output (binary output), the number of fuzzy rules to cover all input conditions is in the order of thousands! This is not only prohibitive from the point of view of the kowledge engineer but also imposes limitations on the reactivity of the fuzzy controller. Hence we have partitioned the ten ultrasonic sensors into five regions (Figure 16) corresponding to the physical geometry of the agent. These are: `right corner`, `left corner`, `right`, `front`, and `left`. In order to account for the beam angle of the sonars and the fuzzy nature of the five regions we adopt sensor *overlapping* across neighboring regions. The sensor arrangement and the overlapping perceptual regions are shown in Figure 16.

Similar to us, Reignier [17] also partitions the sensors into regions. Yet in order to determine the depth value of each region he only relies on a single sensor (the sensor that has minimum depth) . However, from our experience with the sonars of the Labmate the quantitative value of an individual sensor is not reliable at all. Instead, all sensors in a region must be taken into consideration

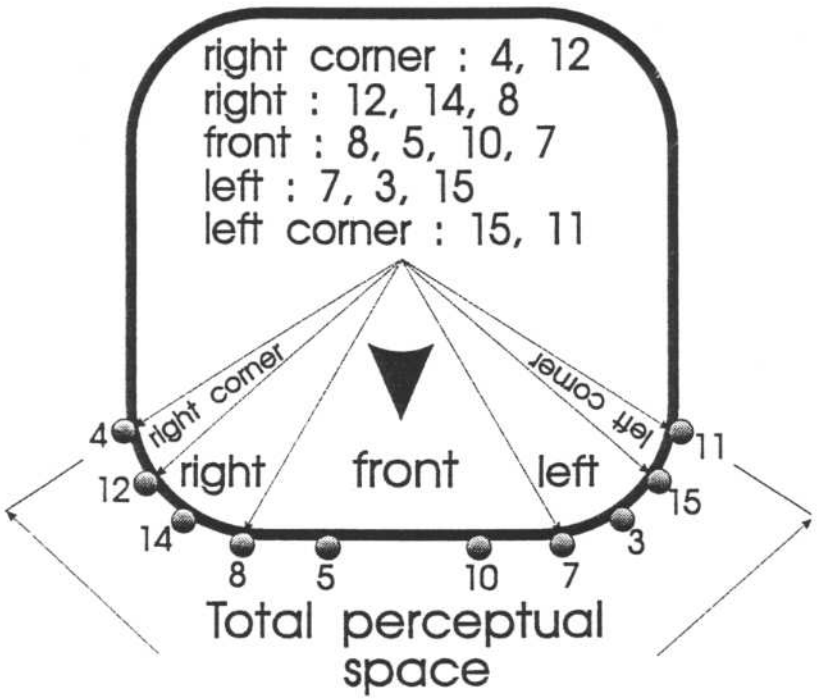


Figure 16: Overlapping perceptual regions

and filtering over time is advisable as well. In [20] we therefore partitioned the sensors into regions and refired the sonars multiple times at each perceptual cycle to gather data from which we estimated the depth of each region. However, multiple firing introduces noise in the system through sensor *crosstalk* and is time consuming.

Rather than relying on the quantitaive value of an individual sensor or firing the sonars repeatedly, we nowadays employ a cascade of two filters (Figure 17) and a sliding window of size 3 to hold the present and the past two measurement profiles. Following partitioning, the sensor values of each region are passed through a median filter (spatial filtering), which gives as output a single measurement of the depth of a region. The median filter estimates the current *measured depth* $\mathcal{Z}_{j,t}$ of a region j covered by N_j sensors using

$$\mathcal{Z}_{j,t} = median(\mathcal{S}^j_{1,t}, \mathcal{S}^j_{2,t}, \ldots, \mathcal{S}^j_{N_j,t}) \ . \tag{33}$$

where $\mathcal{S}^j_{i,t}$, $i = 1 \ldots N_j$, is the reading at time t of sensor i located in region j, N_j the number of sensors in region j and $\mathcal{Z}_{j,t}$, $j = 1 \ldots 5$ the measured depth of region j at time t. However, the measured depth $\mathcal{Z}_{j,t}$ is still noisy and too unreliable for reactive control[5]. Therefore, a Kalman filter is employed to further process the measured depth.

5.2 Kalman Filter Formulation

The proposed Kalman filter operates on the present and past measurement profiles $(\mathcal{Z}_{j,t}, \ldots, \mathcal{Z}_{j,t-n})$, stacked in the sliding window, to estimate the current true

[5]When these values are used to generate control commands the robot is seen moving arbitrary.

depth $\mathcal{D}_{j,t}$ of a region j. To avoid the influence of very past measurements on the present estimate only a limited window size ($n = 2$) is taken. Because a *Bayesian* viewpoint is adopted, we need to select a model for the conditional probability density function (CPDF) of the true depth given the measured depth $\mathcal{P}(\mathcal{D}_j/\mathcal{Z}_j)$ that best fits the data generated by real world. In this paper a Gaussian[6] CPDF is chosen. The main motivations for making this assumption is that the Kalman filter so designed is optimal with respect to virtually any criterion that makes sense [22]. As our viewpoint is Bayesian, we require the filter to propagate the assumed CPDF from some time $t - n$, for some arbitrary n, up to the present time t. Once the CPDF is propagated the optimal estimate is computed using the *maximum likelihood* criterion.

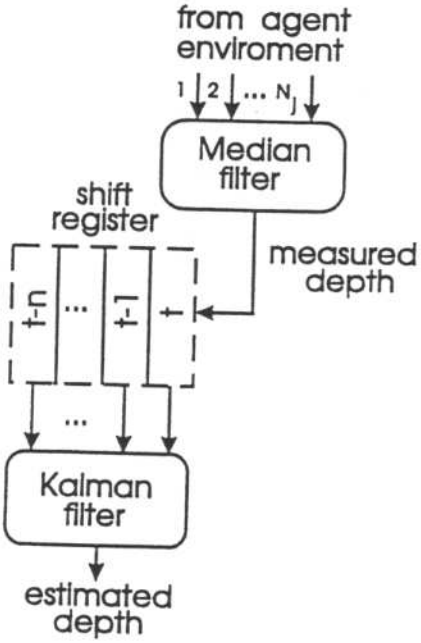


Figure 17: The proposed sensor pre-processor

The Kalman filter algorithm is tailored to suit the agent at hand. To proceed with the algorithm, at each perceptual time t the filters in each region $j = 1 \ldots 5$ are initialized by estimating the parameters of the Gaussian CPDF, mean (μ_j) and variance (σ_j^2). We estimate the mean by equating it with the measured value at time $t - n$ i.e,

$$\mu_{j,0} = \mathcal{Z}_{j,t-n}; \quad j = 1 \ldots 5 \ . \tag{34}$$

and the variance $\sigma_{j,0}^2$ with the measurement variance $\tilde{\sigma}^2$ of the sonars.

To compute the measurement variance, we have picked a sensor at random[7] and placed the sensor in different environments and at different orientations and depths that can be faced by the robot when it is in operation (such as corners, corridors, doors edges, walls, free ways, ...). For all environments and depths, the

[6]There is no mathematical or experimental prove that guarantees a Gaussian noise distribution in ultrasonic sensors.

[7]All the sensors are of the same Polaroid type.

sensor was fired and the true (d_i) and measured (r_i) depths were recorded. After recording $N = 1000$ (d_i, r_i) pairs, the measurement variance $\tilde{\sigma}^2$ is computed as

$$\tilde{\sigma} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(d_i - r_i)^2} = 137mm \ . \tag{35}$$

Yet this value turned out to be too low to represent the actual measurement variance of the sensors when fired one after the other on the moving robot. Hence we multiplied it by a factor of 2.5 to obtain $\tilde{\sigma} \approx 350$. At the beginning of the updating algorithm the statistical variance is set to this measurement variance, i.e,

$$\sigma_{j,0}^2 = \tilde{\sigma}^2 \quad j = 1 \ldots 5 \ . \tag{36}$$

With (34) and (35) the CPDF of each region is defined. The next step is to propagate this CPDF forward up to time t. Inherently our system is dynamic, i.e. agent position and hence sensor values change with time. Therefore, the dynamic Kalman filter best suits our scenario, yet this filter requires a model for the rate of change of the sonar return. For a situated agent, this change depends among other things on: the speed and rotation of the robot, the direction of motion, the environment and its acoustic properties, the dynamic properties of each sensor, the position of the sensors on the robot and the frequency of sensor crosstalk. Looking at the parameters involved, it is extremely difficult to come up with a clean mathematical model of the form of (37) and (38) for the dynamic filter:

$$\dot{\mathcal{X}}(t) = \mathcal{A}(t)\mathcal{X}(t) + \mathcal{B}(t)\mathcal{U}(t) + \mathcal{V}(t) \ . \tag{37}$$

$$\mathcal{Z}(t) = \mathcal{C}(t)\mathcal{X}(t) + \mathcal{W}(t) \ . \tag{38}$$

Here matrices $\mathcal{A}(t)$, $\mathcal{B}(t)$ and $\mathcal{C}(t)$ are system time varying coefficients incorporating all the above mentioned parameters, vectors $\mathcal{X}(t)$ and $\mathcal{Z}(t)$ are estimated and measured depths respectively, and $\mathcal{V}(t)$ and $\mathcal{W}(t)$ are system and measurement noises respectively.

Because of lack of the above system coefficients, a linear recursive Kalman filter is employed, and the CPDF is updated only at discrete time steps, when a measurement value is available. At each update step $i = 1, \ldots, n$ and for any perceptual region $j = 1 \ldots 5$, the updating algorithm is given by :

- compute Kalman gain:

$$\mathcal{K}_{j,i} = \frac{\sigma_{j,i-1}^2}{\sigma_{j,i-1}^2 + \tilde{\sigma}^2} \ . \tag{39}$$

- update mean:

$$\mu_{j,i} = \mu_{j,i-1} + \mathcal{K}_{j,i}(\mu_{j,i-1} - \mathcal{Z}_{j,t-n+i}) \ . \tag{40}$$

- update variance:

$$\sigma^2_{j,i} = (1 - \mathcal{K}_{j,i})\sigma^2_{j,i-1} \ . \tag{41}$$

Figure 18 shows how the parameters of the CPDF, μ_j and σ^2_j, vary at each update. At the last update, we have the CPDF of the estimated depth given the present and the past two measured values, $\mathcal{P}(\mathcal{D}_{j,t}/\mathcal{Z}_{j,t-2}, \mathcal{Z}_{j,t-1}, \mathcal{Z}_{j,t})$. Once this CPDF is determined, the *maximum likelihood* criterion is used to extract the best estimate from the CPDF, i.e,

$$\begin{aligned} \mathcal{D}_{j,t} &= max\ \mathcal{P}(\mathcal{D}_{j,t}/(\mathcal{Z}_{j,t-2}, \mathcal{Z}_{j,t-1}, \mathcal{Z}_{j,t})) \\ &= \mu_{j,2}; \quad j = 1 \ldots 5 \ . \end{aligned} \tag{42}$$

We have implemented a separate Kalman filter according to Figure 17, (34), (35) and (39–42) for each region. Taken together they define our *sensor preprocessing* stage.

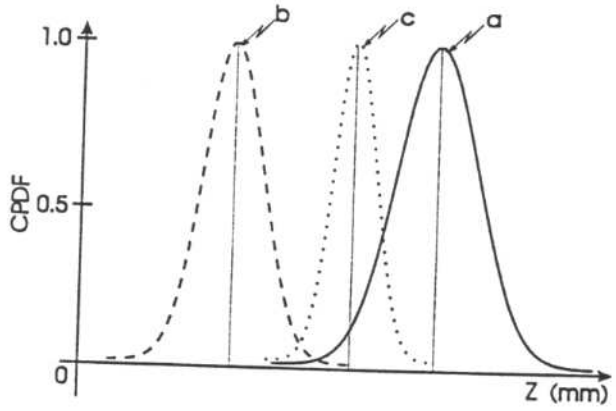


Figure 18: Variation of CPDF of $\mathcal{D}_j$ based on (a) $\mathcal{Z}_{j,t-2}$, (b) $\mathcal{Z}_{j,t-2}, \mathcal{Z}_{j,t-1}$, (c) $\mathcal{Z}_{j,t-2}, \mathcal{Z}_{j,t-1}, \mathcal{Z}_{j,t}$

5.3 Experimental Results

In order to test the preprocessing stage, the robot was placed at a distance of $2m$ in front of a wall. After firing all the sonars the robot is set to move against the wall at a constant velocity. While it was moving, we kept on recording the readings of the sonars in the front regions (sensors $5, 7, 8$ and 10) until the robot approached the wall. Afterwards we applied our as well as Reignier's pre-processing algorithm to the data gathered. Figure 19 shows a plot of both results over time. Clearly, our pre-processing (broken line) provides the better estimate. In particular, notice how the Kalman filter holds (sustains) the depth estimate at a relatively high value with only little oscillations when the robot is far from the wall.

Apart from this off-line test, the proposed sensor pre-processing has been used with an FRBF controller on the actual TRC robot. It enabled the robot to move in our office environment and to pass even narrow doors.

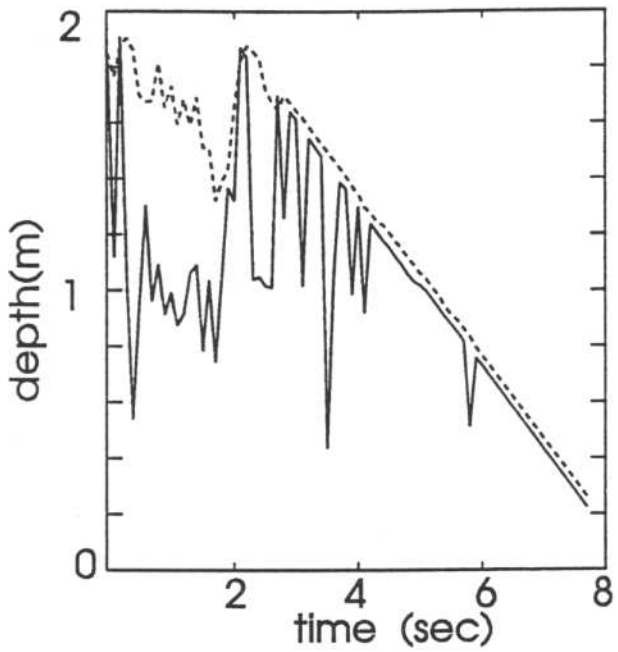


Figure 19: Performance test of the proposed sensor preprocessor

References

[1] J. Bruske and G. Sommer, "Dynamic cell structure learns perfectly topology preserving map," *Neural Computation*, vol. 7, no. 4, pp. 845–865, 1995.

[2] D. Nauck, F. Klawonn, and R. Kruse, *Neuronale Netze und Fuzzy-Systeme.* Braunschweig/Wiesbaden: Vieweg, 1994.

[3] B. Kosko, *Neural Networks And Fuzzy Systems.* Englewood Cliffs, New Jersey: Prentice Hall, 1992.

[4] J. Jang and C. Sun, "Functional equivalence between radial basis function networks and fuzzy inference systems," *IEEE Transactions on Neural Networks*, vol. 4, no. 1, pp. 156–158, 1993.

[5] I. Ahrns, "Ultraschallbasierte navigation und adaptive hindernisvermeidung eines autonomen mobilen roboters," Master's thesis, Inst. f. Inf. u. Prakt. Math., CAU zu Kiel, 1996.

[6] T. Martinetz and K. Schulten, "Topology representing networks," in *Neural Networks*, vol. 7, pp. 505–522, 1994.

[7] I. Ahrns, J. Bruske, and G. Sommer, "On-line learning with dynamic cell stuctures," in *ICANN'95*, pp. 141–146, 1995.

[8] M. J. Mataric, "Integration of representation into goal-driven behavior-based robots," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 3, pp. 304–312, 1992.

[9] J. Buhmann, W. Burgard, *et al.*, "The mobile robot rhino," *AI Magazine*, vol. 16, no. 1, 1995.

[10] J. Bruske, I. Ahrns, and G. Sommer, "An integrated architecture for learning of reactive behaviors based on dynamic cell structures," *Robotics and Autonomous Systems*, 1997. in press.

[11] R. S. Sutton, "Reinforcement learning architectures for animats," in *Proc. of the First International Conference on Simulation of Adaptive Behavior*, pp. 288–296, MIT Press, 1990.

[12] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.

[13] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[14] T. Takeuchi, Y. Nagai, and N. Enomoto, "Fuzzy control of a mobile robot for obstacle avoidance," *Information Science*, vol. 45, pp. 231–248, 1988.

[15] K. T. Song and J. C. Tai, "Fuzzy navigation of a mobile robot," in *Proceeding of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Raleigh), pp. 621–627, 1992.

[16] C. C. Jou and N. C. Wang, "Training a fuzzy controller to back up an autonomous vehicle," in *IEEE International Conference on Neural networks*, (San Francisco), pp. 923–928, 1993.

[17] P. Reignier, "Fuzzy logic techniques for mobile robot obstacle avoidance," *Robotics and Autonomous Systems*, vol. 12, pp. 143–153, 1994.

[18] A. Saffiotti, H. Ruspini, and K. Konolige, *Using Fuzzy Logic for Mobile Robot Control.* Kluwer Academic, 1997. Forthcoming.

[19] D. A. Pomerleau, *Neural Network Perception for Mobile Robot Guidance.* Kluwer, 1993. Norwell, MA.

[20] G. Hailu, "Distributed fuzzy and neural network based navigational behaviours," Technical Lab. Report H/1996, CAU, Cognitive Systems Laboratory, 1996.

[21] S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," *Artificial Intelligence*, vol. 55, pp. 311–365, 1992.

[22] P. S. Maybeck, *The Kalman Filter: An Introduction to Concepts*, pp. 194–204. Springer Verlag, 1994.