Part III SPIN

Chapter 7 SPIN Structure

The *SPIN* system (from **S**patial **P**erception to **I**dentification with **N**eural networks) is suggested as an example for the application-related modelling process as introduced in chapter 2. The main purpose is object-recognition, where the emphasis is on an adaptive recognition process rather than on a specific object-recognition-task. The symbolic part of the recognition-process is intentionally excluded. Symbolic object recognition is on the one hand an interesting and still strongly investigated field, but on the other hand too far from the mobile robot related problems. The principal aspects here are sub-symbolic abstraction in order to enhance the performance of a following symbolic system. The generated symbols (here: surface-clusters) produced as the output of the *SPIN*-structure should be representative for the current environment. The importance for mobile robots is obvious, due to the fact that the application-field here are explicitly unknown environments. Nevertheless such adaptive recognition processes can be used for a wide range of other tasks. The symbols (in contrast to any set of predefined symbols) can be recognized and reconstructed from the flow of input data in a very stable, reliable, and reproducible manner, as will be shown below.

The process is based on the data from a high-precision range finder, applied to follow (i.e. to scan) edges in 3-d. The measuring device is mounted on a mobile robot manoeuvring in a structured and moderate dynamic environment. "Structured" means the existence of a sufficient number of scannable edges and "moderate dynamic" expresses slow changes. The output is an environment model consisting of convex clusters, which are generalized, completed and classified. This output model is the base for the interface to a symbolic (classic) object recognition system (not part of the *SPIN*-project).

The *SPIN*-system is completely simulated, in contrast to the modelling levels discussed above in the *ALICE*-project, where nearly every aspect is tested under real world and realtime conditions. Real world and realtime aspects of *SPIN* are mentioned in chapter 5 as well as in the final conclusions (chapter 13).

7.1. Construction Principles (Philosophies)

The system-design is based on some main principles, which have in common that none of them is an obvious contradiction to biological systems. It is not objected to find the best fitting model for the lower levels of the mammal object-recognition-system, but obviously implausible features should be avoided. This first section contains only philosophies, which are used as guidelines in the *SPIN*-project. The following list should not be considered as definitions but as motivations associated with the introduced keywords.

The actual scope and capabilities are presented in section 7.2 and following chapters.

a. **Realtime Abilities**
The time-correctness, i.e. the timeliness of an information or action is a very critical constraint on a mobile robot. The requested adaptability of the whole system must be related to the most recent situation. Therefore the off-line learning, adaptation or other processes must be avoided or restricted to a mini-

mum, because otherwise the robot has to stop in order to give the internal modelling the change to catch up with realtime, which is a very inconvenient feature and in a large number of applications simply impossible. Thus the selected respectively developed techniques are carefully investigated regarding their realtime abilities.

b. **Adaptability (Learning)**
Building up a really adaptive system means leaving open as much degrees of freedom as possible and useful. So the main part of the "model-base" or "feature-memory" should be able to learn from its environment. Furthermore even the extraction process itself can be adaptive with regard to the current situation.

c. **Unsupervised Learning**
The term "unsupervised" has to be specified in this context. Unsupervised learning is usually assumed when the learning is only triggered by the flow of examples presented to the network. But someone could have selected the examples (as positive or negative examples, as to be presented in a special order, etc. pp.). Is this "selector" not a supervisor?
Here "unsupervised" is used as a synonym for "learning by examples" and the selector of the examples is the outer world, because all the possible input (scanned scene) of a certain kind available from the environment is presented (sooner or later) to the learning-units. There is no instance discriminating between "important" and "unimportant" data from the environment.

d. **Guiding Heuristics**
A kind of "soft" selector (a guiding heuristic) of the current learning-sets is tolerated, when it fulfils some restrictions. A guiding heuristic should prefer some structures, but always in a fair manner, i.e. a certain constellation is tested first but this has no influence to the fact that all the constellations are tested. So all the guiding rules should be formulated as "Try this set first, but don't forget to inspect the other ones afterwards". This kind of heuristics has no influence to the domain of learnable features, but some features are learned quicker than others.

e. **Hierarchical & Symmetric Layers**
One of the major guiding questions, while constructing the *SPIN*-structure, was "What could be the next, most plausible level of abstraction?". So the system was build up bottom-up, without grasping for really object-recognition results at each individual stage. The resulting structure is pipeline-oriented, symmetric regarding each wider step and hierarchical.
The general purpose processes (like classification) should be quite similar or identical at several hierarchical levels. This means, once the system is able to classify geometric figures, it should not be of significant influence, what these geometric elements are at a certain stage.

f. **Feedback**
A strict hierarchy is not as useful as it could be, if the different layers are not connected in both directions. Without this feedback, the layers form a simple pipeline. Two different kinds of feedback are discriminated in the following.

f-1 **Error Feedback**
The first form of feedback are back-propagated error-messages, which may take effect over multiple layers in the counter-direction of the data-flow. This error-flow corrects decisions on lower abstraction levels. Thus there is no need for finding the right answer always or immediately. The most likely way can be chosen at a certain stage, knowing that there is another instance, which will correct this decision later, if this way was wrong.

f-2 **Point-of-interest Feedback**
The other frequently used form of feedback is point-of-interest generation, i.e. here: guiding of lower-level units based on higher level knowledge. At the start-up-time of the whole system, each unit is either waiting for input, or show some activity, initiated only by the local "instinct". Later, while the flow of information has reached all components, the activities of lower-level units are more and more triggered by higher-level components.

g. **Parallelism**
The pipeline-organisation of the *SPIN*-system leads to the extensive usage of parallelism. Additionally, while taking a closer look at the components, there are some more possibilities of parallelism regarding the single neural nets.

g-1 **Parallelism at a coarse granularity**
This first form of parallelism is oriented at the abstraction components, i.e. there might be an own processor or group of processors for each stage in the abstraction pipeline. The flow of data as well as the error- and the point-of-interest-feedback is buffered between the processing elements and there are no control packets in the counter directions documenting the processing of the information (flow-control). The stages represented by one processing element could be any kind of feature extractor, classification units, etc.

g-2 **Parallelism at a fine granularity**
Implementing neural nets offers an obvious chance for using parallelism, but in most cases a special hardware will become necessary. Due to the fact that the available hardware-components are not really suitable for a wide range of connectionist methods yet (as applied here), the realisation of the fine granularity parallelism is not part of the current *SPIN*-project.

7.2. SPIN Structure

The *SPIN*-system as it is currently implemented is shown in figure 6. The structure can be divided into the following sub-systems:

- **The Edge-Surface Detector**
  This module is the source of all information gathered from the environment. It consists of a freely-controllable laser-range-finder, where several edge-following strategies are applied, in order to scan 3-d edges in the current environment (see [Schäfer93] for details). The assumption applied is that the current environment contains a significant number of scannable edges, or that the scannable edges are sufficient to build a consistent model of forms or objects. This is usually the case in indoor and especially in office environments, but it is of course an assumption that can be wrong applied to arbitrary

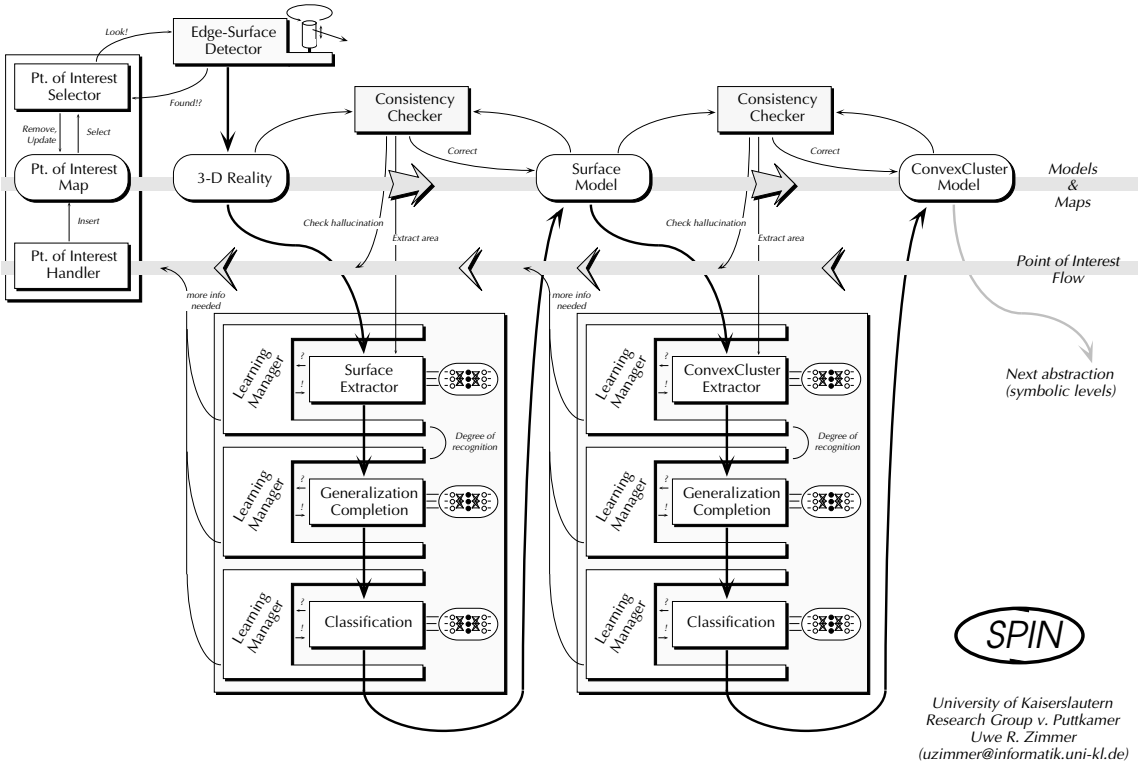

Figure 6: SPIN main structure

mobile robot tasks. The other critical assumption is that the accuracy of such a laser-range-finder must be in a range that enables the following components to calculate curvatures of these edges. This is possible with state-of-the-art range-measuring-devices, but it can be still critical in specific situations, as for example measuring distances to specific surfaces or materials in flat angles. The speed of edge-scanning depends widely on the mechanical construction for the beam-steering, but this will not influence the principal analysis of such a system as it is shown here. Nevertheless this component will (in the opinion of the author) be the most critical one, regarding a practical application (see chapter 13 for a discussion).

- **Models & Maps**
  The unsupervised building of adequate models and maps from the environment is the main purpose of the *SPIN*-project. Thus the bar of models in figure 6 is the central structure. The idea is to earn a certain abstraction from the left to right side, where the left-most model ("**Point of interest map**") plays a special role (see below). The first and most immediate model is the "**3-d Reality**" consisting only of edges scanned in the current environment. Special dynamic data-structures are applied to make the model efficient (see [Schäfer93] for details). Edges forming a closed polygon are already represented as surfaces here together with the connectivity information between neighbouring surfaces. The next model ("**Surface Model**") consists of completed, generalized and classified surfaces extracted from the 3-d reality model. Significantly different information can be stored here according to the world models in the several processing steps leading to the surface model (see below for details). The difference between the models at different abstraction levels leads to a first possibility to trigger or steer the focus of attention in this system. The **consistency checkers** between the several stages detect the differences and try to get an explanation for them. First there can be something missing in the higher model. Thus a component responsible for extracting features for the pipeline processing towards the next abstraction level is triggered to check this feature. Second there can be something missing in the lower model but be represented in the higher one. Obviously this is an hallucination of one of the processing stages, but it can be a valid or a wrong one. This is checked by a wish for a detailed scan ("check hallucination") at this spacial point in the reality (if possible, i.e. for example if the relevant perspective is currently not occluded). Finally, contradictions (which can be detected) between the models are solved by eliminating them in the higher model. The third model ("**Convex Cluster Model**") is the straightforward expansion to the next higher abstraction step. It is based on the surface model and represents completed, generalized, and classified clusters of surfaces, which are common in the current environment. The interface to the following (symbolic object-recognition-) systems has to label these elements and construct meaningful objects or features out of this model.

- **Abstraction pipelines**
  Between each two adjacent models on different abstraction levels exists a complete pipeline of processing steps, necessary for the abstraction process. In the current version there are two (very similar) abstraction-pipelines to generate the surface-model and the convex-cluster-model. In the first stage the feature that should be used as the basic element on the next abstraction level has to be extracted. This selection has a major influence on the whole process, because all the following "learning-by-examples" pipeline stages are very sensitive to the "boot-strap-phase" (i.e. the first elements to be processed have a wide impact) and sensitive to the further order of presented examples. Therefore some pre-defined knowledge is used at this extraction-stage in the form of a neuro-fuzzy system described in chapter 3. The **generalization** and **completion** process is integrated in one pipeline stage, where the best matching (learned) complete surface resp. surface-cluster from the (distributed and adaptive) model-base is produced as an output. The completion-error measured at this stage ("Degree of recognition") is employed as an error-feedback for the feature extraction stage, which can optimize the selection process according to this signal, after a certain time-delay. The last pipeline stage is the **classification** part, which clusters the stream of surfaces respectively surface-clusters. The three pipeline stages are implemented as separate processes, where each stage is approximated using several techniques of learning-by-examples. The employed techniques will be critically compared during the following chapters.

- **Point-of-Interest System**
  Each component in the SPIN-structure is allowed to formulate wishes regarding the exploration process, sequence or strategy of the environment. This could include requests like "Gather more information at a specific area in the environment" or "Check a specific feature respectively hallucination found during the abstraction process". All these requests are collected in a "**point-of-interest flow**" and stored in a "**point-of-interest map**". The strategies employed for selecting the actual scan area are called "focus of attention" or "visual search" strategies and will be discussed in chapter 8. In the cur-

rent version the form of requests is limited to an evaluation of specific scan-processes, i.e. each component can give a "vote" for the efficiency of an executed edge scanning.

7.3. Edge, Surface Representations

The main data-structure applied in the *SPIN*-project is based on geometric information extracted from scanned edges. In order to be position- and orientation-invariant, a border-curvature-representation of the surfaces is chosen. The curvatures are calculated by projecting the border (at each border point) on two orthogonal two-dimensional planes and regarding the 2-d-curvatures on these planes (called the surface- and the segment-curvature). The orientations of these planes are defined at each border point by the orientation of the local border tangent and the local surface normal: Both planes have to include the border tangent and one of them (the plane to determine the surface-curvature) has to include the surface normal.

Assuming a parametric form *{x(t), y(t)}* of a 2-d border fragment, where $t \in [0, 1]$. The 2-d curvature *k* of this fragment can then be defined analytically by:

$$k = \frac{y''}{1 + y'^2} \tag{3}$$

where

$$y' = \frac{dy}{dx} \text{ and } y'' = \frac{d^2y}{dx^2} \tag{4}$$

Defining

$$\dot{x} = \frac{dx}{dt};\ \ddot{x} = \frac{d^2x}{dt^2};\ \dot{y} = \frac{dy}{dt};\ \text{and } \ddot{y} = \frac{d^2y}{dt^2} \tag{5}$$

y' respectively y'' can be expressed by:

$$y' = \frac{\dot{y}}{\dot{x}} \text{ and } y'' = \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{\dot{x}^3} \tag{6}$$

resulting in the final, parametric form of the curvature *k*:

$$k = \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{(\dot{x}^2 + \dot{y}^2)^{3/2}} \tag{7}$$

The curvature *k* in the form of (7) is very sensitive to noise in the analysed curve. Applying Gaussian kernels $g(t, \sigma)$:

$$g(t, \sigma) = \frac{e^{-\frac{t^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}} \tag{8}$$

to the input curve, the parametric function $X(t, \sigma)$ can be expressed by the convolution of *x(t)* with the Gaussian kernel:

$$X(t, \sigma) = x(t) \cdot g(t, \sigma) = \int_{-\infty}^{\infty} x(u) \frac{e^{-\frac{(t-u)^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}} \tag{9}$$

Analogous $Y(t, \sigma)$ can be derived from *y(t)*. The smoothed curvature k_σ is then given by:

$$k_\sigma = \frac{\dot{X}\ddot{Y} - \dot{Y}\ddot{X}}{(\dot{X}^2 + \dot{Y}^2)^{3/2}} \tag{10}$$

where

$$\dot{X} = x(t) \cdot \frac{\partial g(t, \sigma)}{\partial t};\ \ddot{X} = x(t) \cdot \frac{\partial^2 g(t, \sigma)}{\partial t^2};\ \dot{Y} = y(t) \cdot \frac{\partial g(t, \sigma)}{\partial t};\ \text{and } \ddot{Y} = y(t) \cdot \frac{\partial^2 g(t, \sigma)}{\partial t^2} \tag{11}$$

Due to the fact that the underlying sensor-samples, on which all these calculations are based on are very sparse, these plane-orientations together with the resulting curvatures are approximated with simple discrete techniques [Schäfer93], [Stecher94] or generated directly in simulations [Keuchel92], [Trundt94].

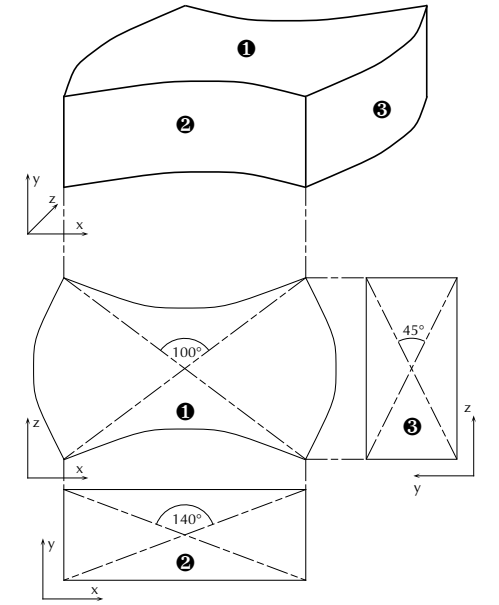

Figure 7: 3-d sample object

Each surface is then described by the concatenation of two vectors; one consisting of the surface-curvatures at m_1 equidistant points along a whole cycle along the border and another consisting of the segment-curvatures at m_2 equidistant points along the same distance. The maximum number of curvature sample points along the border is limited by the sampling rate of the range-finder, which has scanned the surface edges.

This surface vector-representation is based on the assumption that the form of a surface is sufficiently described by the curvatures at its border, i.e. any characteristics e.g. at the middle of the surface are not detected at all. The motivation for this assumption is first observed and described by Attneave in [Attneave54], postulating that information on the shape of a curve is concentrated at dominant points with high curva-

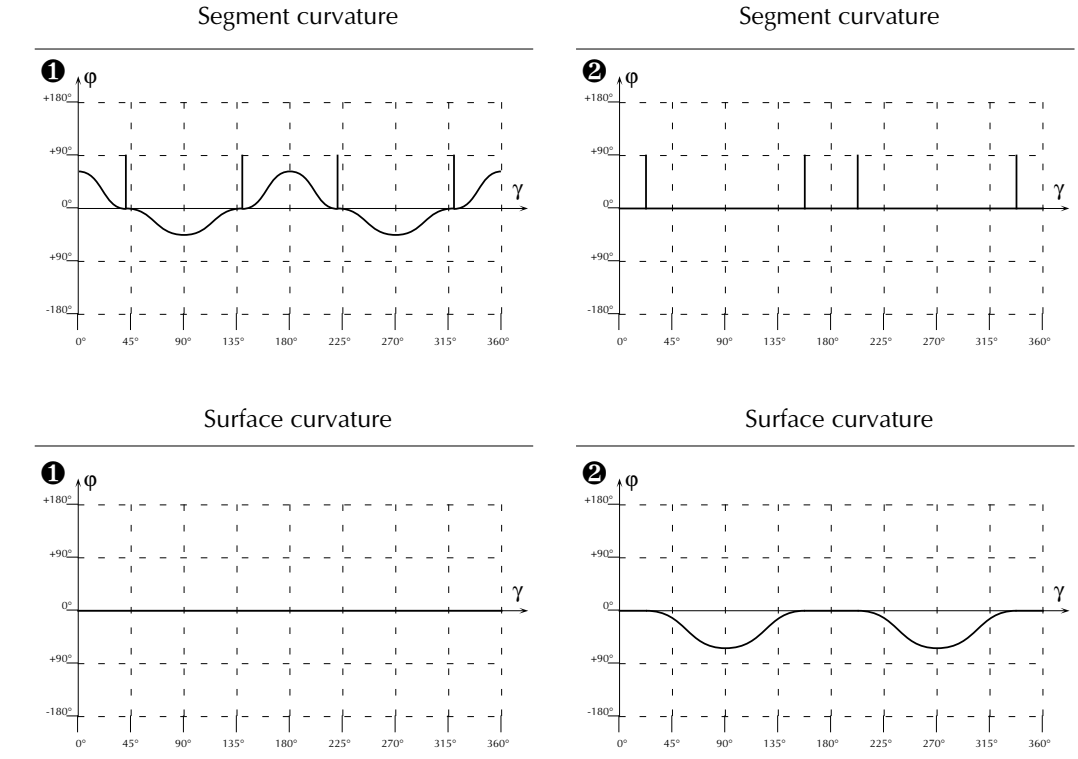

Figure 8: Segment and surface curvatures

ture. Later on, Hubel and Wiesel ([Hubel68], [Hubel65], [Hubel62], [Hubel59]) discovered structures in the receptive fields of the cat's and monkey's visual cortex, based on contrasts (and specific filters) instead of absolute values, which make the employment of (higher) derivatives in visual processing biologically plausible.

As an example of this technique, two surfaces of the 3-d object from figure 7 are represented with their border-curvatures as shown in figure 8.

A discussion of other classical methods like Fourier descriptors or invariant moments for the description of 2-d curves can be found in [Bebis91]. For the special *SPIN* sub-task concerning surface completion, frequency coding (Fourier transformation) has been applied also (see [Grund93]).

7.4. Discussed Modules (the following chapters)

Four characteristic modules out of the *SPIN*-system will be carefully discussed in the following chapters. Each of these modules represents another connectionist concept, but all of them fulfil the requirements postulated above. Some of the discussed techniques (visual search and surface fusion) are especially designed for the *SPIN*-project and published in this thesis for the first time (beside the already published articles mentioned in the beginning of this thesis). One further neural concept (Adaptive Resonance Theory) is well-known but applied to specific problems in the sub-symbolic object recognition field (surface-cluster classification). Finally, one module (surface classification) represents a significant expansion of an already introduced technique. The discussed *SPIN*-components are mentioned according to their order in the processing-pipeline (figure 6).

Visual Search

The visual search method presented in chapter 8 suggests a new framework for selective attention strategies. A new clustering technique, especially adapted for realtime representations of spacial densities of samples is employed for the selective attention sub-system. In order to be able to interrupt ongoing search procedures, a fast reinforced learning component is applied driven by the feedback from the following consuming components in the processing-pipeline.

Surface Fusion

The process of producing “acceptable” surface representations from scratch on, before an adaptation to the current environment could take place, is a very critical task. Therefore a neuro-fuzzy-system is proposed here, which composes the possibility of formulating coarse a-priori knowledge together with a precise adaptation phase during the application. The neuro-fuzzy-decision-system (called *SPIN-NFDS* in the following) demonstrates very promising features, which will be discussed and compared to other recent approaches in chapter 3.

Surface Classification

Multiple approaches regarding the surface classification problem have been tested in the *SPIN*-project. The first tested technique is the GAL (Grow-And-Learn) model by Alpaydin [Alpaydin91] applied to the *SPIN* surface classification task in the thesis from Sigmann [Sigmann93]. A generalization-tree approach based on unsupervised competitive neural network is discussed in the thesis of Klapper [Klapper93]. Finally, “problem dependent cell structures” (or “dynamic unsupervised clustering techniques”) based on the ideas presented in [Fritzke91a], are applied and specifically expanded regarding the surface-classification task. This work will be shown and discussed in chapter 4.

Surface-Cluster Classification

Classification of surface-clusters is very similar to the classification of single surfaces. The main differences are the feature vector and the “natural” expectations. Test-runs have been done by Wagner regarding the set of useful and significant features describing surface-clusters in a compact and fixed-length form [WagnerM93]. The several feature vectors are tested with the dynamic unsupervised clustering-network presented in [Zimmer93b]. The simulations shown in chapter 11 concentrate on the ART 2 respectively on the ART-2A approach proposed by Carpenter and Grossberg (e.g. [Carpenter91b] or [Carpenter87a]).

7.5. Other SPIN-Investigations

The *SPIN*-project is discussed on the base of selected (key-) modules. Thus the reader is asked to refer to the following list of other investigations in the context of *SPIN* for any further information. The list is related to the order of processing in the *SPIN*-pipeline:

- Generation of an 3-d world model and adequate control of a laser range finder: [Schäfer93]
- Efficient geometric world modelling: [Stecher94]
- Selective attention strategies for active surface completion: [Nölke94]
- The extraction of surfaces: [Stecher92]
- A neural fuzzy decision system: [Bruske93b]
- Completion of 3-d surfaces with multi-layer backpropagation networks and frequency coding: [Grund93]
- Generalization and completion of surfaces with multi-layer backpropagation networks: [Trundt94]
- Classification of surfaces on the base of generalization trees and unsupervised neural networks: [Klapper93]
- Dynamic surface classifications on the base of "Grow and Learn" (GAL) from Alpaydin [Alpaydin91]: [Sigmann93]
- Dynamic classification of surfaces with growing cell structures: [Keuchel92]
- Dynamic classification of surface-clusters on the base of Adaptive Resonance Theory (ART): [Keuchel94]
- Dynamic classification of surface-clusters with problem dependent cell structures: [WagnerM93]

All these works were guided and supervised by the author.

Chapter 8 Visual Search

Visual Search has been investigated by many researchers inspired by the biological fact, that the sensory elements on the mammal retina are not equably distributed. Therefore the focus of attention (the area of the retina with the highest density of sensory elements) has to be directed in a way, gathering data efficiently according to certain criteria.

The work discussed in this chapter concentrates on applying a laser range finder instead of a silicon retina. The laser range finder is maximal focused at any time, but therefore a low-resolution total-scene-image, available with camera-like devices from scratch on, cannot be used here. By adapting a couple of algorithms, the edge-scanning module steering the laser range finder is able to trace a detected edge. Based on the scanned data, two questions have to be answered. First: "Should the actual (edge-) scanning be interrupted in order to give another area of interest a chance of being investigated?" and second: "Where to start a new edge-scanning, after being interrupted?".

These two decision-problems might be solved by a range of decision systems. The correctness or efficiency of the decisions depends widely on the actual environment and the underlying rule-base may not be well initialized with a-priori knowledge. In section 8.1 the author will present a version of a reinforced decision system together with an overall scheme for efficient control of highly focused devices.

Another aspect of visual search consists of local search strategies in order to gather relevant data from a certain (small) area efficiently. When applying standard control theory to the tracing component of the edge-scanner, the system is usually lost, whenever smaller or larger "gaps" have to be overcome along an edge. These gaps can occur rather often, because any disturbance of the high-precision range-finder, any kind of occlusion, or any edge-part with "uncooperative" angles will result in misses. For detailed descriptions of such problems please refer to [Schäfer93]. An approach to overcome this dilemma is discussed in section 8.2.

The implementation of several reinforcement models together with extensive test environments are performed by Nölke [Nölke94] and Stecher [Stecher94].

8.1. Efficient Visual Search

Visual Search is a dynamic research field, investigated by researchers from many different disciplines. The author will highlight only some main work concerning efficient visual search and applying connectionist techniques (mainly self-organizing feature maps).

Ahmad and Omohundro have presented a multi-feature-map-system being able to search for and to locate target objects with certain features (namely equilateral triangles, [Ahmad90]) in static camera images and (more general) to detect feature correspondences between a camera frame and a model space ([Ahmad91], [Ahmad94]). Marshall ([Marshall93b], [Marshall91a]) has shown general techniques to handle context-sen-

sitive problems like trajectory tracing, grouping and steering in visual motion perception with the "EXIN"-network.

The visual search system presented here is mainly conceived as a support system for geometric environment abstractions on an autonomous mobile robot[1].

8.1.1. Concepts and Structure

The main structure is based on the decomposition in three main parts as shown in figure 9:

- An (to a certain degree autonomous) **edge-scanner**, which is able to perform 3-d edge-following. Beginning at a given starting point, the edge-scanner finds recursively all edges, which are noticed while following the current edge. The result is the edge/surface map (denoted "3-d reality" in the *SPIN*-overview from section 7.2 on page 59), which represents discrete orientations and curvatures of all scanned edge-fragments.
- Based on the edge/surface-map a long **pipeline of abstraction steps** (up to a generalized, completed and classified cluster representation, which will be used by a symbolic level) follows. The "efficiency"[2] of the actual data-flow is measured at each stage.
- Finally the edge-scanner is controlled by the **focus-of-attention manager**, whose decisions are based on the efficiency feedback from the abstraction pipeline. The control task is divided into two sub-modules: The **area-of-interest generator**, which selects a region in the actual environment where an efficient knowledge accumulation might be likely, and a "disturbance module" called **restlessness generator**, which determines when to choose a new target area.

In the next two sections the author will focus the discussion on these two decision sub-modules.

8.1.2. Area-of-Interest Generator

The area of interest is selected on the basis of statistical information about the environment. The statistical information includes spacial densities of edges, free-space and unknown regions. In order to collect and represent these densities, a special, dynamic self-organizing map with 3-dimensional simplexes (tetrahedrons) is used as the basic structure. The simplexes divides the space in Voronoi regions, attributed with

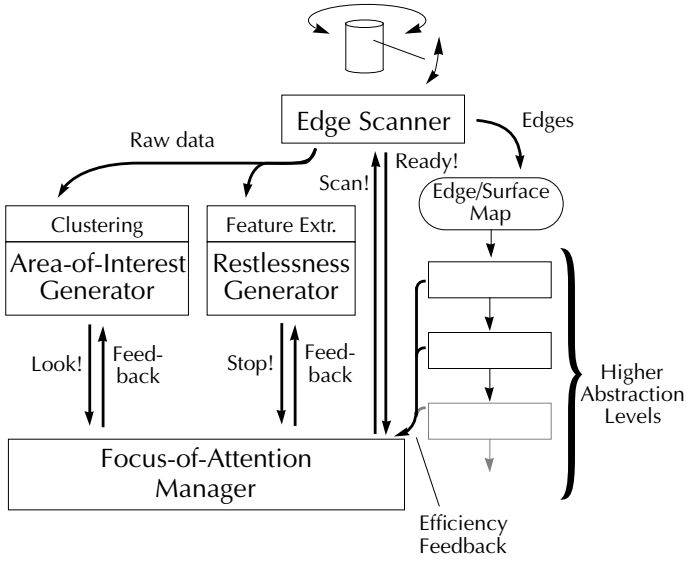

Figure 9: Focus of attention system

1. *see [Zimmer93a], [Zimmer93b] respectively chapter 7 for more information about other parts of the SPIN-project*
2. *a term which has to be determined by each individual abstraction step*

information about the amount of sensor data collected in this region as well as different densities (edges/free-space). The specific attributes attached to each voronoi-region (cell) *c* are:

- The mean quantification error τ_c accumulated with each classification step referring to the cell *c*.
- The mean quantification error τ_c^t accumulated during the last *t* classification steps referring to *c*.
- The mean distance to neighbouring cells F_c as an approximation for the size of the voronoi-space represented by *c*.
- The number of occupied points τ_c^o represented by *c*.
- The number of free-space points τ_c^f represented by *c*.

The network is adapted by replacing single cells with new simplexes or by fusing neighbouring simplexes, according to the following criteria:

Insertion

- $(\tau_c > \lambda) \wedge (F_c > \varepsilon)$; where λ controls the relative cluster-granularity and ε limits the lower size of the cell's voronoi region. This condition is applied to detect situations, where the number of representatives is too low in a specific region.
- $(\tau_c^o > 1) \wedge (\tau_c^f > 1) \wedge (F_c > \varepsilon)$. Each time a cluster has to represent occupied and free space at the same time, the region is split.

The insertion process itself is outlined in figure 10, where a cell representing occupied space is forced to represent free-space also, and is therefore replaced by a new simplex. For clearness, triangles are used as simplexes in a two-dimensional input space, but the reader should keep in mind that these simplexes are usually of a higher dimension.

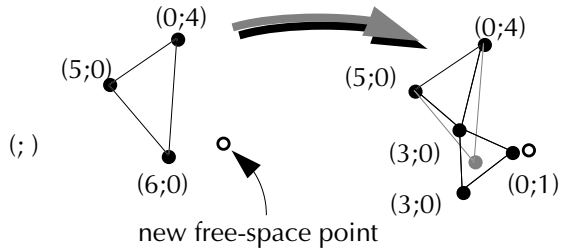

Figure 10: Replacement of a cell with a simplex

Fusion

- Assuming a simplex *S* with cells $c_1, \ldots, c_n$. *S* is fused to a single cell, if:

$$(max_{c_i \in S}(\tau_{c_i}^t) < \tau_\varepsilon) \wedge \left(\left(\sum_{i=1}^{n} \tau_{c_i}^o < 1 \right) \vee \left(\sum_{i=1}^{n} \tau_{c_i}^f < 1 \right) \right) \tag{12}$$

  i.e. the number of updates during the last *t* steps is low and *S* represents only free space or only occupied space. Moreover, in order to keep a structure of simplexes, a re-ordering must take place or the fusion process must be restricted to simplexes with only one connection to the remaining graph. In the current version a re-ordering of any kind is not implemented.

For other insertion/fusion criteria please refer to [Stecher92].

This unsupervised process gives the needed input for the area-of-interest selection. Up to now the decisions are done by simple heuristics (search strategies), without using the efficiency feedback from the higher abstraction levels. In several tests the number of scanned edges/surfaces is significantly increased (in comparison to recursive edge-following) only by applying the clustering together with simple heuristics, but even more can be achieved by using the restless-generator, whose decisions are already based on the efficiency feedback (see next section).

This module is actually a framework for search strategies, which are adapted manually to the needs of the abstraction pipeline. Although the results are already encouraging, it is planned to *select* the search strate-

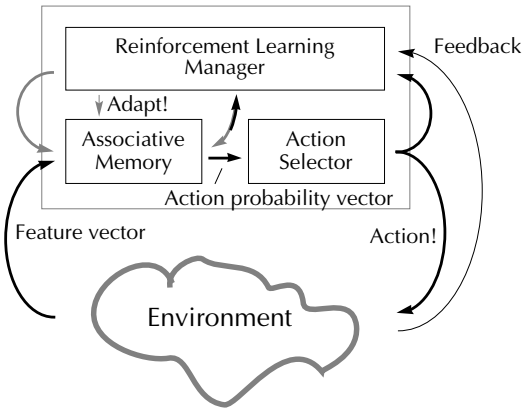

Figure 11: Reinforcement learning

gies by a reinforcement system (as applied in the restlessness generator) as well as a neural fuzzy decision system (*SPIN-NFDS* [Zimmer93a] respectively chapter 3) for a further improvement of this task.

The tests with several search strategies seem to imply that the key issue of our visual search scenario is an adequate clustering rather than a sophisticated search heuristic.

8.1.3. Restlessness Generator

Due to the fact that one edge scan may acquire a lot of time gathering information of low use for a certain application (e.g. scanning walls), a module is introduced, which interrupts the actual scanning in order to give another (perhaps more productive) area in the environment the chance of being explored. In contrast to camera based systems, allowing the usage of the whole (low resolution) scene for this decision, the highly focused laser range finder delivers only information about the currently scanned edge. Therefore the features used for the decision are exclusively extracted from the current flow of laser-measurements:

- The length of the currently scanned edge.
- The sum of the absolute angles between the currently scanned edge-fragment.
- The mean distance between the edge and the background.
- The number of retries during the edge-scanning (the edge may be "lost" during the trace-procedures).

In order to keep the input space small, only these four statistical features are applied, despite the fact that many others could be introduced.

According to the reinforcement learning scheme shown in figure 11, an action probability vector is defined consisting of the different possible actions attributed with the probability this action should be chosen. The actions used here are: "Stop scanning immediately" and a couple of "Try another x elementary measurements, before asking me again". The probability attributes are generated by an associative memory module. A dynamic supervised feature map system is used (Growing cell structures [Fritzke93c]) instead of multilayer backpropagation or cascade correlation, because of the high stability and adaptation speed.

To allow trials with the several strategies, the final action selector determines the action to test by the action probabilities and a random number generator. This prevents that the action with the highest probability is chosen always and therefore that a continuous "gradient decent" will go down the first path chosen randomly at system start time.

The feature vector, the action probability vector together with the selected action, and the efficiency feedback is used to adapt the mapping in the associative memory. The probability of the selected action is increased respectively decreased by a certain factor according to the efficiency feedback and finally the other probabilities are scaled in order to keep the sum of all probabilities one.

The number of found and useful edges is increased in any of our tests applying this module, whereas the amount of improvement depends widely on the actual environment. Therefore the author is limited to qualitative results here.

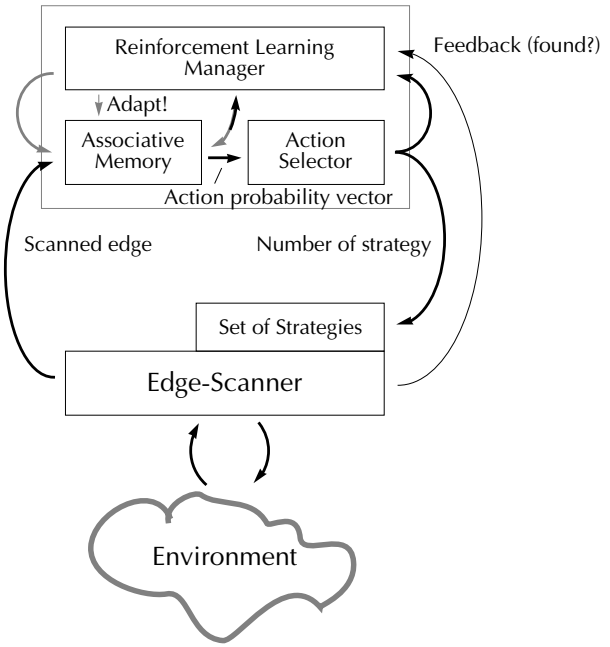

Figure 12: Selective Attention Structure

8.2. Adaptive Surface Completion

Whenever the scanner has lost its track along an edge, and even adaptive control strategies are not able to continue the scan originating from the last detected edge-part, a heuristic is needed to guess a likely continuation in a certain distance. The approach shown here is based on a fixed set of continuation strategies and a reinforced learning system for the selection of the most promising strategy. The general structure is shown in figure 12, where the reinforcement learning component is identical to the system shown in section 8.2. The incompletely scanned edge is represented by its curvatures as introduced in section 7.3. The memory module associates a probability vector consisting of seven entries, each attached to a scan-continuation strategy (i.e. a set of seven strategies is used during the tests). According to the probabilities, one of these strategies is applied in the edge-scanning module. The evaluation can be done immediately and gives a binary feedback, i.e. the strategy has found the lost edge again or not. Due to the actual feedback, the probability for the selected strategy is slightly incremented or decremented and the rest of the vector is scaled to keep the sum of probabilities one. The new probability vector is trained together with the appertaining edge representation. Once again the associative memory is implemented on the base of the supervised growing cell structures [Fritzke93b]. Thus the learning times are mainly determined by the number of steps in the reinforcement phase, i.e. especially not by the adaptation times of the associative memory. Each reinforcement step results in an experiment in the real world, i.e. the number of learning steps is critical in a real world application. On the other hand, there are no distinct learning and execution phases. Therefore the system is applied continuously, where the only change should be that the number of misses is going down continuously in a static environment. This continuous optimization can be shown applying a fixed learning set[3]. The error concerning a single training sample is measured in the following terms:

$$E_\xi = 1 - \left(\sum_{i \in S} \left(p_i(\xi) \cdot f_i(\xi)\right)\right) \tag{13}$$

where ξ is the training sample; S is the set of strategies;
$p_i(\xi)$ is the probability for strategy i applying sample ξ; and the feedback $f_i(\xi_j)$ is given by:

$$f_i(\xi) = \begin{cases} 1 & ;\text{strategy i succeded for sample } \xi \\ 0 & ;\text{strategy i missed for sample } \xi \end{cases} \tag{14}$$

Accordingly the error for a whole learning set T is defined by the mean error:

3. Minor drawbacks during the learning phase stems from the original removal strategy defined in "growing cell structures" [Fritzke93c] and could be easily avoided applying removal strategies shown in [Zimmer93b] and section 4.1.2.

$$E_T = 1 - \left(\frac{1}{|T|} \sum_{j \in T} \sum_{i \in S} \left(p_i(\xi_j) \cdot f_i(\xi_j)\right)\right) \tag{15}$$

The following experiment is based on ten surfaces of different shapes cut at random border positions. Due to the fact that the emphasize is not on sophisticated continuation strategies, but on the selection of these strategies on the base of a reinforced learning system only 2-d surfaces and strategies are applied. The seven used continuation strategies are very simple (e.g. reproduce the last x edge-segments, or look at the middle of the current gap, etc.). The several windows shown in figure 13 represent the reinforcement learning stages after certain numbers of learning steps, where the small circles in the upper right corner above each surface fragment represent (with the dark parts) the current error-rate of the system regarding that surface fragment. In the beginning the error-rates are rather high (50-90%). Some of the surfaces can be attached to a good continuation strategy during the first learning steps, where others need several trials with different strategies, before a adequate one can be identified after 5000-7000 learning steps. But still there are some surfaces (one in the example from figure 13), where no working continuation strategy could be found in the current set. The only and obvious way to overcome this weakness is to add further algorithms to the set of strategies. But the focus of this experiment is the continuous decrease of the global error i.e. especially not the elimination of each individual error in all configurations. The global error in the shown test is below 10% in the final stage, where some surfaces are better than 3% and one is at 60% error rate. The shown ex-

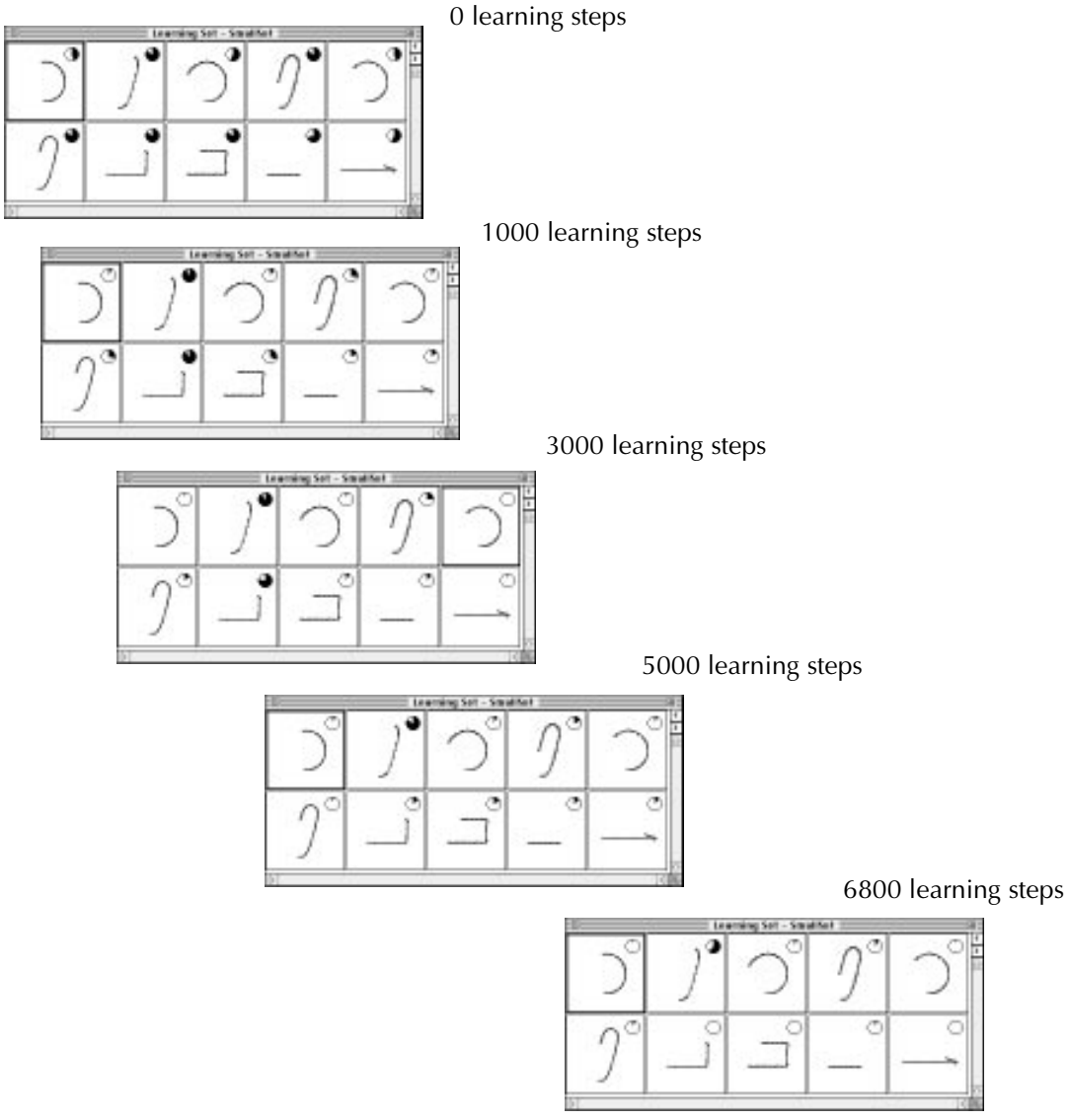

Figure 13: Reinforcement learning on 10 surface fragments

periment is representative for the family of simple surfaces (like Polygons and Ellipses) cut at random border positions. The mean number of learning steps for a test-set with randomly cut surfaces is 4280 with a standard derivation of 882, where the tests are stopped when the global error rate is below 10%. The derivation is descended from the indeterminism in the action selection module as well as in the order of presentation and positions of gaps of the surfaces.

8.3. Real World Remarks

The components discussed in this model are very closely related respectively connected to the "real world (interface)". Moreover and more specifically, the adaptations are done in close cooperation with the immediate reactions from the environment. The actual environment plays therefore an important role for the methods presented in this chapter. On the other hand all the tests are done on the base of simulations, due to the fact that nor an adequate range finder neither a steering mechanism were available. For this reason, the relevance of the components is questionable, whereas the key issue is not only the real world test itself, but also the definition of representative test-environments. The author could show a qualitative, positive impact of the discussed methods in simple, simulated environments. Depending on the configuration, this impact is very encouraging, but the reader should keep in mind, that the relevance has to be discussed again with regard of more realistic sensor models or specific environments. The author has currently no concrete idea about these critical configurations, but their existence could not be excluded.

Some discussed parts contain large degrees of freedom (especially where sets of algorithms or strategies can are handled) and can thus be used as a general framework for visual search strategies applied to highly focused devices. The actual (realistic) sensor model will have a strong impact on the (reduction of the) degrees of freedom for an applicable visual search system.

Chapter 9

Surface Fusion

The task to be discussed in this chapter may be characterized in short by the question: "Given the representation of two surface fragments, decide whether they originally belong together (i.e. form one surface) or not?" (figure 14). This problem appears as a part of the surface extraction process described above. The surface representation used here is the common representation as defined in section 7.3. Because of the fact, that the surface extraction is the first stage in the surface-processing-pipeline and all further components are only trained by the flow of presented surface examples, implausible surfaces should be avoided here at all times. This does not imply that learning techniques cannot be used here, but that a technique has to be found using preset knowledge as a well defined initial state of our decision system.

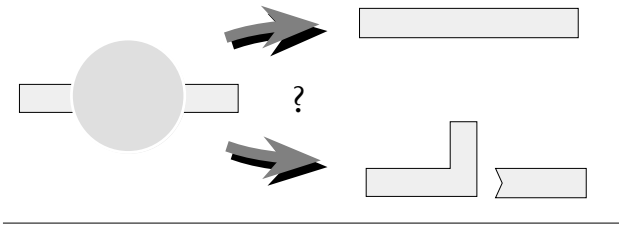

Figure 14: The surface fragmentation problem

As the basis for the decision process, features need to be defined, which have to be extracted from the two surface fragments. The optimal decision function, which is able to find the correct answer in any case, cannot be found for principal reasons, like the fact, that the decision process is highly context-dependent, whereas the decision component has no access to the complete context in a real world. Therefore the complete set of features, which would be sufficient for an optimal decision system, cannot be constructed explicitly. But a basic-set of features for a well defined statistically based system will be defined in the following. For an illustration of the following features see figure 15.

- **Real-edge-curvature differences *($recd_a$, $recd_b$)***
  A finite part of the edges (e_{1a}, e_{2a}) is defined on both sides of the obscured area. The mean-curvature is calculated at each edge-part and the difference between them is used as the first feature ($recd_a$). The other curvature difference ($recd_b$) is calculated analogous for the second pair of edge-parts (e_{1b}, e_{2b}).

- **Obscured-edge-curvature difference *(oecd)***
  The mean-curvature along the obscured edge of one surface fragment is either calculated on the basis of measurements along this virtual edge o_1, or by interpolating the surface-orientations at the end parts of the real edges (e_{1a}, e_{1b}). The difference between these curvatures at the obscured edges o_1 and o_2 is used as another measurement for similarity at the critical surface parts.

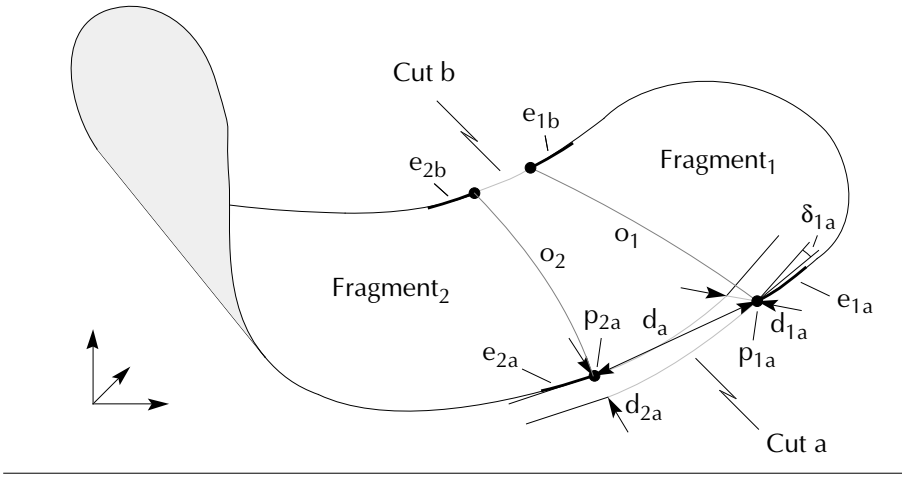

Figure 15: Feature Extraction

- **Extrapolation distances** (d_{1a}, d_{1b}, d_{2a}, d_{2b})
  An extrapolation of the edge-end parts e_{1a} and e_{2a} is determined and the minimal distance between the extrapolation of e_{1a} and the endpoint p_{2a} respectively the distance between extrapolation of e_{2a} and the endpoint p_{1a} is calculated (resulting in d_{1a} and d_{2a}). Analogous the distances d_{1b} and d_{2b} are calculated at cut b.

- **Extrapolation-angle differences** (δ_{1a}, δ_{1b}, δ_{2a}, δ_{2b})
  The orientation of the extrapolation at the point of minimal distance to the referring endpoint is calculated and compared to the edge-orientation at this end-point itself. All four extrapolations are handled by this procedure resulting in four extrapolation-angle differences δ_{1a}, δ_{1b}, δ_{2a}, δ_{2b}.

The feature extraction procedures as defined above are not directly applicable. The number of orientation-measurements per edge-length is quite low, because of the complex and time-consuming techniques needed to trace 3d-edges with a laser-range-finder. So the useful kinds of extrapolations are limited to simple methods. For the discrete formulations please refer to [Bruske93b].

9.1. Learning as well as Preset Knowledge

As a motivation for the kind of decision systems presented later in this chapter, the main problems and advantages of knowledge-based and learning system in our context will be discussed first. The learning system discussed here is the well known multi-layer-backpropagation-system (MLBP) and the knowledge-based system is a fuzzy logic system, introduced in detail in the next chapter. The fuzzy logic system is employed as a non-linear method, mapping exact input to exact output information, i.e. especially not as a method handling non-quantitative, linguistic information.

The central problem using the MLBP-technique is the missing a-priori ability to find acceptable decisions from scratch on. This is the main argument, why such techniques cannot be applied to the surface-fusion task. The generally long learning periods of MLBP-systems makes this problem even worse. On the other hand fuzzy logic systems have full abilities at any time and they do not need any learning periods at all, but at least in our surface-fusion task they reach only a poor level of correctness in comparison to a well adapted MLBP-system.

Based on the two main advantages (ad-hoc abilities of the fuzzy logic system and low error-rates of the MLBP-system) of the two considered techniques, a combination of both will be constructed, called a Neural-Fuzzy-Decision System (NFDS). The central idea is to derive a multi-layer-feed-forward network as an implementation of a predefined fuzzy-logic system and to adapt this rough decision system by back-propagation-learning on examples from the current environment. In the best case the resulting decision-system will offer ad-hoc abilities and the possibility of reaching lower error-rates in a short learning period (table 1). It will be shown in section 9.3 on page 137 that the simulations generally indicate this optimal behaviour.

Learning Systems (Multi-Layer-BP)	Neural-Fuzzy-Decision-System	Knowledge-Based (Fuzzy Logic)
No a-priori abilities	Ad-hoc abilities	Ad-hoc abilities
Long learning periods	Shorter learning periods	No learning periods
Low error-rates (after learning)	Low error-rates (after learning)	**High** error-rates

Table 1: A comparison between learning and knowledge-based systems

9.2. Known Neural-Fuzzy-Combination Techniques

This chapter gives a short survey of some existing Neural-Fuzzy-Decision-Systems[4]. In order to find a common terminology for the several approaches some definitions are given in the first part. Finally two of the most recent technologies (NARA and NNFCDS) are presented and discussed in this section. Other models, not discussed here can be found in [LeeCC91], [Jang92a], [Giles93], and [FuHC94].

9.2.1. Definitions

The well known concept of a **multi-layer-feed-forward network** with adaptation by **backpropagation** do not need a further formalisation. Please refer for a short and precise introduction to [Hertz91] pp. 115-130 or for a more general presentation to [Rumelhart85] vol. 1 pp. 318-362.

A quite different impression may be got, when trying to find a common notation for the family of rule-based systems called "**Fuzzy Logic**". There are a lot of hard to match terminologies, so the author is forced to give a small set of definitions in order to make a common base available for the following models.

9.2.1-1 Triangular Norms

In the following several triangular norms will be employed, which have in common that they are all two-placed functions from $[0, 1] \times [0, 1]$ to $[0, 1]$.

Definition 1: The operator "$\otimes$" represents one of the following triangular norms:

$$x \wedge y = min\{x, y\} \qquad \textbf{(intersection)} \tag{16}$$

$$x \cdot y = xy \qquad \textbf{(algebraic product)} \tag{17}$$

$$x \overline{\cdot} y = max\{0, x + y - 1\} \qquad \textbf{(bounded product)} \tag{18}$$

$$x \overline{\cap} y = \begin{cases} x & ; y = 1 \\ y & ; x = 1 \\ 0 & ; x, y < 1 \end{cases} \qquad \textbf{(drastic product)} \tag{19}$$

where

$$(x \overline{\cap} y) \le (x \overline{\cdot} y) \le (x \cdot y) \le (x \wedge y) \tag{20}$$

4. also denoted as "Neural-Networks designed on Approximate Reasoning", "Adapted-Network-Based Fuzzy Inference System", "Neural-Network-Based Fuzzy Logic Control and Decision System", "Back-Propagation Fuzzy System", "Rule-Based Network-Architecture" and several others.

Definition 2: The operator "⊕" represents one of the following triangular co-norms:

$$x \vee y = max\{x, y\} \quad \textbf{(union)} \tag{21}$$

$$x \dotplus y = x + y - xy \quad \textbf{(algebraic sum)} \tag{22}$$

$$x \overline{\dotplus} y = min\{1, x + y\} \quad \textbf{(bounded sum)} \tag{23}$$

$$x \overline{\cup} y = \begin{cases} x & ; y = 0 \\ y & ; x = 0 \\ 1 & ; x, y > 0 \end{cases} \quad \textbf{(drastic sum)} \tag{24}$$

where

$$(x \vee y) \le (x \dotplus y) \le (x \overline{\dotplus} y) \le (x \overline{\cup} y) \tag{25}$$

The triangular norms "⊗" are employed for defining conjunctions in approximate reasoning, while the triangular co-norms "⊕" serve the same role for fuzzy disjunctions.

9.2.1-2 Fuzzy Sets and Set Theoretic Operations

Definition 3: A **fuzzy set** F in a universe of discourse U is characterized by a **membership function** M_F: $U \rightarrow [0,1]$. A fuzzy set F in U may be represented as a set of ordered pairs of a generic element u and its grade of membership:

$$F = \{[u, M_F(u)] \mid u \in U\}. \tag{26}$$

Definition 4: A **fuzzy number** in a continuous universe U is a fuzzy set F in U which is **normal** and **convex**:

$$max\ \{M_F(u), u \in U\} = 1 \text{ (normal)} \tag{27}$$

$$M_F(\lambda u_1 + (1 - \lambda)u_2) \ge min\ \{M_F(u_1), M_F(u_2)\}\ \forall\ u_1, u_2 \in U; 0 \le \lambda \le 1 \text{ (convex)} \tag{28}$$

Definition 5: The **support of a fuzzy set** F is the set of all $u \in U$ such that $M_F(u) > 0$.

Definition 6: A **fuzzy singleton** is a fuzzy set whose support is a single $u \in U$ with $M_F(u) = 1$.

Let A, B be two fuzzy sets in U, and $A_1, \ldots, A_n$ fuzzy sets in $U_1, \ldots, U_n$ in the following.

Definition 7: The **union** $A \cup B$ is characterized by the membership function $M_{A \cup B}$ with:

$$M_{A \cup B}(u) = max\ \{M_A(u), M_B(u)\},\ \forall\ u \in U \tag{29}$$

Definition 8: The **intersection** $A \cap B$ is characterized by the membership function $M_{A \cap B}$ with:

$$M_{A \cap B}(u) = min\ \{M_A(u), M_B(u)\} \text{ or} \tag{30}$$

$$M_{A \cap B}(u) = M_A(u) \cdot M_B(u),\ \forall\ u \in U \tag{31}$$

Definition 9: The **complement** $\bar{A}$ is characterized by the membership function $M_{\bar{A}}$ with:

$$M_{\bar{A}}(u) = 1 - M_A(u) \tag{32}$$

Definition 10: The **cartesian product** $A_1 \times \ldots \times A_n$ in the product space $U_1 \times \ldots \times U_n$ is characterized by the membership function $M_{A_1 \times \ldots \times A_{1n}}$ with:

$$M_{A_1 \times \ldots \times A_{1n}}(u_1, \ldots, u_n) = min\ \{M_{A_1}(u_1), \ldots, M_{A_n}(u_n)\} \text{ or} \tag{33}$$

$$M_{A_1 \times \ldots \times A_{1n}}(u_1, \ldots, u_n) = M_{A_1}(u_1) \cdot \ldots \cdot M_{A_n}(u_n),\ \forall\ (u_1, \ldots, u_n) \in U_1 \times \ldots \times U_n \tag{34}$$

Definition 11: An **n-ary fuzzy relation** R is a fuzzy set in $U_1 \times \ldots \times U_n$ and is expressed as:

$$R_{U_1 \times \ldots \times U_n} = \{[(u_1, \ldots, u_n), M_R(u_1, \ldots u_n)] \mid (u_1, \ldots, u_n) \in U_1 \times \ldots \times U_n\} \tag{35}$$

Let R be a fuzzy relation in $U \times V$ and S a fuzzy relation in $V \times W$.

Definition 12: A **sup-star composition** of R and S is a fuzzy relation denoted by $R \bullet S$ with:

$$R \bullet S = \{[(u, w),\ sup_{v \in V}\ \{M_R(u, v) \otimes M_S(v, w)\}] \mid u \in U, w \in W\} \tag{36}$$

where "⊗" can be any operator in the class of triangular norms (see definition 1).

9.2.1-3 Fuzzy Implications

In order to define a complete fuzzy decision system, several techniques implementing fuzzy-implications have to be introduced. Since the introduction of fuzzy-logic (see e.g. chapter "[Zadeh73]"), nearly 40 distinct fuzzy implication relations have been proposed for different applications (see chapter "[LeeCC90b]" for most of the references). Most of the approaches may be classified into the following three categories. Let A respectively B be a fuzzy set in U respectively in V.

Definition 13: The **fuzzy conjunction** is defined by:

$$(A \rightarrow B)_{\otimes} = \{[(u,v), M_A(u) \otimes M_B(v)] \mid u \in U, v \in V\} \tag{37}$$

where "⊗" can be any operator in the class of triangular norms (see definition 1).

Definition 14: The **fuzzy disjunction** is defined by:

$$(A \rightarrow B)_{\oplus} = \{[(u,v), M_A(u) \oplus M_B(v)] \mid u \in U, v \in V\} \tag{38}$$

where "⊕" can be any operator in the class of triangular co-norms (see definition 2).

Definition 15: The **fuzzy implications** are classified into five groups:

$A \rightarrow B =$

(**Material implication**) $= \{[(u,v), M_{\bar{A}}(u) \oplus M_B(v)] \mid u \in U, v \in V\}$ (39)

(**Propositional calculus**) $= \{[(u,v), M_{\bar{A}}(u) \oplus (M_A(u) \otimes M_B(v))] \mid u \in U, v \in V\}$ (40)

(**Extended propositional calculus**) $= \{[(u,v), (M_{\bar{A}}(u) \otimes M_{\bar{B}}(v)) \oplus M_B(v)] \mid u \in U, v \in V\}$ (41)

(**Generalized modus ponens**) $= \{[(u,v), sup_{c \in [0,1]} \{c \otimes M_A(u) \leq M_B(v)\}] \mid u \in U, v \in V\}$ (42)

(**Generalized modus tollens**) $= \{[(u,v), inf_{c \in [0,1]} \{c \oplus M_B(v) \leq M_A(u)\}] \mid u \in U, v \in V\}$ (43)

Based of this definitions a large number of fuzzy implications may be generated by employing different triangular norms and co-norm. Some of the commonly used implications are listed below.

(**Mini-operation rule**, Mamdani) $= \{[(u,v), M_A(u) \wedge M_B(v)] \mid u \in U, v \in V\}$ (44)

(**Product-operation rule**, Larsen) $= \{[(u,v), M_A(u) \cdot M_B(v)] \mid u \in U, v \in V\}$ (45)

(**Arithmetic rule**, Zadeh) $= \{[(u,v), M_{\bar{A}}(u) \overline{+} M_B(v)] \mid u \in U, v \in V\}$ (46)

(**Maxmin rule**, Zadeh) $= \{[(u,v), M_{\bar{A}}(u) \vee (M_A(u) \wedge M_B(v))] \mid u \in U, v \in V\}$ (47)

The rules introduced by Mamdani and Larsen will be referred below, in order to classify the different types of rule evaluation.

9.2.1-4 Approximate Reasoning

Definition 16: A **linguistic variable** is characterized by a quadruple *(X, T(X), U, M(X))* in which X is the **name of the variable**; $T(X) = \{T_X^1, \ldots, T_X^k\}$ is the **term set** of X, that is the set of **linguistic values** of X with each linguistic value corresponding to a fuzzy number defined on U (or $dom(X) = U$); $M = \{M_X^1, \ldots, M_X^k\}$ is the set of membership functions characterizing these fuzzy numbers.

Definition 17: A **proposition** (syntactical form: "<linguistic variable> is < linguistic value>") has the semantic: "Instantiate the linguistic variable to the fuzzy number associated with the linguistic value".

The fuzzy implication inference rule used here is a form of the **generalized modus ponens** (**gmp**):

premise 1:	X is A'
premise 2:	if X is A then Y is B
consequence:	Y is B'

where A, A', B, B' are fuzzy sets and X, Y are linguistic variables. Premise 2 is a **fuzzy rule**, where the intersection $A \cap A'$ can be regarded as the "degree of fulfilment" of the rule's antecedent, influencing the actual

relation between B and B' according to the following. Writing the premises of the gmp without the linguistic variables leads to (R is a fuzzy relation in $U \times V$)

premise 1:	$\{[u, M_{A'}(u)] \mid u \in U\}$
premise 2:	$\{[(u, v), M_R(u, v)] \mid u \in U, v \in V\}$

Definition 18: The **sup-star compositional rule of inference** asserts that the fuzzy set B' in V is given by

$$B' = A' \bullet R \tag{48}$$

so the consequence may be written as

conseq.:	$\{[v, sup_{u \in U} \{M_{A'}(u) \otimes M_R(u, v)\}] \mid v \in V\}$

Four different kinds of compositional operators may be found in the literature. They are produced by replacing the "⊗" operator by one of the triangular norms from definition 1.

- **sup-min** operator (Zadeh, 1973)
- **sup-product** operator (Kaufmann, 1975)
- **sup-bounded**-product operator (Mizumoto, 1981)
- **sup-drastic**-product operator (Mizumoto, 1981)

In most fuzzy logic controllers the *sup-min* or *sup-product* operators are employed for their simplicity and computational efficiency.

9.2.1-5 Fuzzy Decision System (FDS)[5]

Based on the fuzzy logic foundations above and according to a general scheme for a FDS in figure 16 five basic elements "**membership functions**", "**rule base**", "**fuzzyfication**", "**inference engine**" and "**defuzzyfication**" have to be formalized.

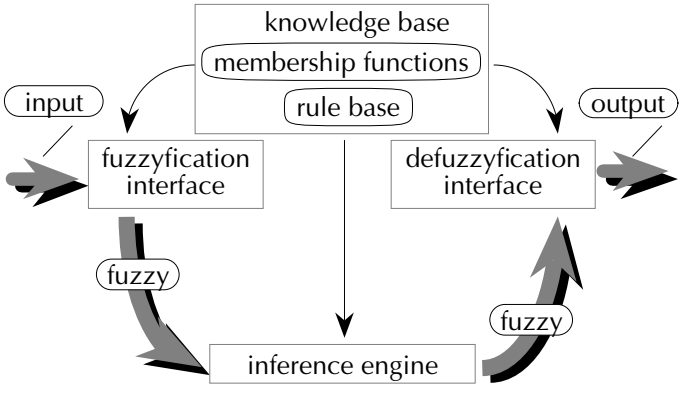

Figure 16: Fuzzy decision system (structure)

Membership Functions

Common membership functions are sigmoid- or gaussian-functions as well as simple linear functions concatenated to form triangle or trapezoid-shaped functions. The applicability of the linguistic term-sets of the linguistic variables are extensively determined by these membership functions. Generally the selection and parameterisation is done on the basis of intuition and partial understanding of underlying physical or other constraints. On the other hand most of the following Neural-Fuzzy-Decision-Systems are able to adapt the functions according to an optimization criteria.

5. also denoted as "fuzzy inference system", "fuzzy associative memory", "fuzzy controller" and several others depending on the context.

Rule Base

Beside the definition of the membership functions, the rule base is the second element to express expert knowledge. The rule base is usually of the form:

if {a set of conditions is satisfied} *then* {a set of consequences is to be inferred}
else
if {another set of conditions is satisfied} *then* {another set of consequences is to be inferred}
else
…

Since the propositions making up the antecedents and the consequences of this *if-then-else* construct are associated with terms from linguistic variables, they are referred to as **linguistic rules**. Several **sentence connectives** like *and* or *else* are linking propositions to an antecedent and fuzzy rules to a complete rule base. There is a huge number of approaches implementing the sentence connectives and the fuzzy implication (relation).

Fuzzyfication

In most fuzzy decision systems non-fuzzy input data are mapped to fuzzy sets by treating them as fuzzy singletons: If $u_0 \in U$ is a non-fuzzy input data and X is the corresponding linguistic variable with $dom(X) = U$ and $T(X) = \{T_x^1, \ldots, T_x^k\}$, the input is fuzzyfied to the fuzzy sets T_x^i by setting the corresponding membership functions:

$$M_x^i(u) = \begin{cases} 1 \;; if\; u = u_0 \\ 0 \;; if\; u \neq u_0 \end{cases} \quad (1 \leq i \leq k) \tag{49}$$

The usual evaluation of the inputs as a fuzzy comparison in the antecedents (e.g. *ActualPressure* is *high*) may now be calculated by the fuzzy intersection like introduced in definition 8 respectively equations (30) and (31). Thus the resulting membership functions evaluate to:

$$M_{x \cap T_x^i}^i(u) = \begin{cases} M_{T_x^i}(u_0) \;; if\; u = u_0 \\ 0 \;; if\; u \neq u_0 \end{cases} \quad (1 \leq i \leq k) \tag{50}$$

Inference Engine

The inference mechanism used here is generally more rudimentary than that used in typical expert systems, since the consequent of a rule does not interfere the antecedent of another. This kind of evaluation may be called "one-level-forward data-driven inference" and is predestined to fast and deterministic controllers.

In a multiple-input-multiple-output (MIMO) system the rule-base may be formulated like:

$$R = \{R^1, \ldots, R^n\} \tag{51}$$

with

$$R^i = \textbf{if}\ ((x_1\ \textbf{is}\ A_{1_i})\ \textbf{and … and}\ (x_k\ \textbf{is}\ A_{k_i}))\ \textbf{then}\ (y_1\ \textbf{is}\ B_{1_i}, \ldots, y_q\ \textbf{is}\ B_{q_i}) \tag{52}$$

The independence of the fuzzy sets in the consequences allows to limit the following discussion to multiple-input-single-output (MISO) systems without less of generality. Considering the general form of a MISO system in the case of a two input/single output system below.

Input: X is A' and Y is B'
R_1: if X is A_1 and Y is B_1 then Z is C_1 else
R_2: if X is A_2 and Y is B_2 then Z is C_2 else
…
R_n: if X is A_n and Y is B_n then Z is C_n else
Output: Z is C'

where X, Y, Z are linguistic variables and A_i, B_i, C_i are linguistic values in the universe of discourse U, V, W.

The fuzzy rule R_i is being implemented as a fuzzy implication (fuzzy relation) characterized by the membership-function:

$$M_{R_i}(u, v, w) = M_{A_i \textbf{ and } B_i \to C_i}(u, v, w) \tag{53}$$

The consequence C' is deduced from the sup-star compositional rule of inference employing the above definitions of fuzzy implications.

Lemma 1: The sup-min operator "•" and the else-connective "∪" applying the inputs A', B' to the rule-set are commutative:

$$C' = (A', B') \bullet \bigcup_{i=1}^{n} R_i = \bigcup_{i=1}^{n} (A', B') \bullet R_i = \bigcup_{i=1}^{n} C'_i \tag{54}$$

Due to lemma 1, it is allowed to evaluate the rules individually, before the final result C' is composed from the results C'_i from each rule R_i. For the proofs (which are straight forward) of this and the following lemma please refer to [LeeCC90b].

Lemma 2: Applying the intersection "∧"or the algebraic product "·" for the sentence connective "and" and the sup-min operator for the individual rule evaluation results in:

$$(A', B') \bullet (A_i \textbf{ and } B_i \to C) = [A' \bullet (A_i \to C_i)] \cap [B' \bullet (B_i \to C_i)] \text{ if } M_{A_i \times B_i} = M_{A_i} \wedge M_{B_i} \tag{55}$$

$$(A', B') \bullet (A_i \textbf{ and } B_i \to C) = [A' \bullet (A_i \to C_i)]\,[B' \bullet (B_i \to C_i)] \text{ if } M_{A_i \times B_i} = M_{A_i} \cdot M_{B_i} \tag{56}$$

Therefore the influence of the inputs to a single rules can be evaluated independently. Equivalently lemma 1' and lemma 2' may been shown using the "sup-product" instead of the "sup-min"-operator.

Since the inputs are fuzzy singletons ($A' = u_0$, $B' = v_0$), the "degree of fulfilment" α of the antecedents may be calculated simply by:

$$\hat{\alpha}_i = M_{A_i}(u_0) \wedge M_{B_i}(v_0) \text{ respectively } \dot{\alpha}_i = M_{A_i}(u_0) \cdot M_{B_i}(v_0) \tag{57}$$

$\hat{\alpha}_i$ and $\dot{\alpha}_i$ are simply denoted as α_i in the following. The set C' derived from the resulting sets of each rule may now be characterized by its membership function:

$$M_{C'} = \bigcup_{i=1}^{n} \alpha_i \otimes M_{C_i} \tag{58}$$

Depending on the choice for the implementation of "⊗" and other more structural differences, four different kind of fuzzy reasoning have to be distinguished.

a. **First type**: *Mamdani's Minimum Operation rule as a fuzzy implication function*
This first type of fuzzy reasoning uses

$$M_{C'} = \bigcup_{i=1}^{n} \alpha_i \wedge M_{C_i} \tag{59}$$

which means, that $M_{C'}$ pointwise defined by:

$$M_{C'}(w) = (\alpha_1 \wedge M_{C_1}(w)) \vee \ldots \vee (\alpha_n \wedge M_{C_n}(w)) \tag{60}$$

b. **Second type**: *Larsen's Product Operation rule as a fuzzy implication function*
The type of fuzzy reasoning is completely identical to the first one, despite the fact, that the algebraic product is used instead of the intersection in the implications.

c. **Third type**: *Tsukamoto's Method with linguistic terms as monotonic membership functions*
This method is used as a speed-up technique for fuzzy reasoning of type 1. All membership functions are forced to be monotonic. Thus the results of the individual rules are simply calculated by:

$$\beta_i = C_i(\alpha_i) \text{ where } \alpha_i = M_{A_i}(u_0) \otimes M_{B_i}(v_0) \tag{61}$$

The defuzzyfication step is then given by a weighted combination:

$$z_0 = \frac{\sum_{i=1}^{n} \beta_i \alpha_i}{\sum_{i=1}^{n} \beta_i} \tag{62}$$

Notice that the membership functions of the fuzzy sets A_i and B_i have not to be monotonic in the context discussed so far.

d. **Fourth type**: *The consequence of a rule is a function of the linguistic input variables*
In this last type of fuzzy reasoning a different structure for the rules itself is being chosen. The i-th rule in the rule-base is formulated in the form:

R_i: if x_1 is A_i and … and x_k is K_i then $z = f_i(x_1, \ldots, x_k)$

The α_is are calculated as usual, so that the crisp results from each rule are given by

$$\beta_i = f_i(\alpha_i) \text{ where } \alpha_i = M_{A_i}(u_0) \otimes M_{B_i}(v_0) \otimes \ldots \tag{63}$$

and the output value is calculated by a weighted sum.

$$z_0 = \frac{\sum_{i=1}^{n} \alpha_i f_i(x_1, \ldots, x_k)}{\sum_{i=1}^{n} \alpha_i} \tag{64}$$

For a detailed description of this model and its applications please refer to [Takagi85].

Defuzzyfication

This last step is usually required, because most of the applications need crisp outputs. The applied techniques depend widely on the application domain, but some basic methods should be mentioned. Generally the defuzzyfication should aim to achieve a crisp output, which represents the fuzzy output optimal.

a. Mean of Maximum (**MoM**)
The mean of maximum method generates an output which represents the mean value of all $w_i \in W$ with $M_C(w_i)$ is a local maximum of the output fuzzy set $\{ [w, M_C(w)] \mid w \in W \}$.

$$z_0 = \frac{\sum_{i=1}^{n} w_i}{l} \text{ with } l \text{ equals the number of local maxima.} \tag{65}$$

b. Centre of Area (**CoA**)
The centre of area method generates a centre of gravity in the possibility distribution of the fuzzy output set. The exact value in a continuous universe is:

$$z_0 = \frac{\int M(w)\, w \, \mathrm{d}w}{\int M(w) \, \mathrm{d}w} \tag{66}$$

In most implementations only a discrete approximation is possible or applicable. But despite the implementation details, this method is the most used and often most useful technique.

For a literature survey with a lot of references on this topic refer to [LeeCC90b].

9.2.2. NARA

NARA (Neural-Networks designed on Approximate Reasoning Architecture) has been introduced and refined by Takagi in his papers [Takagi92a], [Takagi92b], [Takagi92c], [Takagi91] and [Takagi90]. According to our classification NARA represents a “type 4” reasoning system.

Architecture

Neural networks are employed to compute the rule antecedents α_i (one network NN_{mem} for all α_i) and the rule consequences $f_i(\alpha_i)$ (one network NN_i per consequence function $f_i(\alpha_i)$). Finally the output is computed as usual in type four reasoning systems by a weighted sum. Therefore the NARA system is able to learn (or adapt) its consequence functions as well as its membership functions for the antecedents (membership functions of the individual input terms are not represented explicitly). The architecture may be employed recursively, namely by replacing a consequent network by a complete "sub-NARA" structure.

Training

Assuming a training data set $E = \{ (x, y) \mid x \equiv \text{Input vector}, y \equiv \text{Output vector} \}$ the learning process as proposed by Takagi may be structured in five steps.

a. **Clustering of the training set**
The input set has to be clustered into r Clusters C^i, which corresponds to the r rules, which will be employed. The clustering is done on the basis of the input vectors and according to the application in mind, i.e. widely on experience, than on universal algorithms.

b. **Training of the NN_{mem}-network**
The network NN_{mem} is being trained in a supervised manner in order to distinguish the r antecedents of the rules, i.e. the network should produce the following output

$$\alpha_i = \begin{cases} 1 & ; x \in C^i \\ 0 & ; \text{otherwise} \end{cases} \quad ; \text{where } x \text{ is the actual input vector} \tag{67}$$

c. **Training of the r NN_i-networks**
Each network NN_i is being trained to perform the consequence function of the i-th rule, represented by the training pairs $(x, y) \in C^i$. The supervised training procedure has to be stopped in order to prevent overfitting, i.e. to keep the ability to generalize at a certain degree of an acceptable error level.

d. **Pruning of the input vector**
In order to prevent highly correlated elements in the input vector, the above training-steps are repeated without a certain element x_j from the input vector x. If the achieved error-level is not significantly higher than before, the element will be removed. The test is being done for all the elements of the input vector.

e. **Refining (adding sub-nets recursively)**
If there are subnets NN_i with intolerable large training errors, they will be replaced by a complete sub-NARA architecture and the above learning-steps (incl. this) are completely applied to this sub-net. Notice that the clustering technique from step *a* has to be refined, in order to get a more detailed clustering.

Discussion

NARA offers an universal way to approximate complex functions, by partitioning the input space into distinct clusters, where the output is being computed by a weighted sum over the consequence functions. The NARA structure may be interpreted as an implementation of a fuzzy decision system of type 4, but the a-priori-knowledge is only involved via the sample input-/output-pairs and distance measurement applied in the clustering technique (respectively the clustering technique itself), i.e. not via explicit rules or predefined membership functions. Depending on the chosen neural network architecture, NARA is able to learn arbitrary shaped cluster-borders and complex functions. The main disadvantage, as seen by the author, is the limitation to enter a-priori-knowledge only via implicit ways (by using a specific network structure and learning technique or via a distance measurement or clustering technique), i.e. the usual way to formulate knowledge in rules and membership functions cannot be applied here. On the other side NARA offers an universal frame for function approximation techniques using input space segmentation.

9.2.3. NNFCDS

The second discussed model is NNFCDS introduced by Lin and Lee in [LinC91]. The proposed system can be classified as a type 1 fuzzy reasoning system, where the extraction of rules from data is included.

Architecture

NNFCDS consists of a five layer neural network, implementing the transformation from crisp input to crisp output data. The NNFCDS network architecture is shown in figure 17, where the individual layers can be characterized as follows:

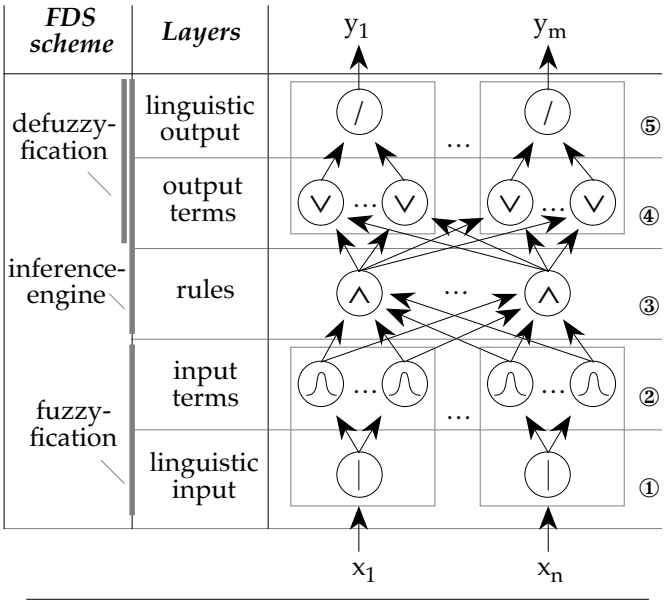

Figure 17: NNFCDS network topology

a. **Layer 1: Linguistic Input**
The crisp input values are spread to the next layer, where the sets of linguistic terms attached to each input variable are implemented. The link to the next layer is thus not fully interconnected.

b. **Layer 2: Input Terms**
Gaussians are proposed by Lin and Lee for the input membership functions. Thus this second layer evaluates these gaussians for each term out of the input term sets. The connections to the next layer may represent any combination of terms in the input sets, but is usually limited according to a pruning procedure introduced below. The parameters (μ_{ij}, σ_{ij}) from the input gaussians are interpreted as the weights between layer one and two.

c. **Layer 3: Rules**
The rule evaluation (the fuzzy "and"-connective) is implemented by a *min*-operator. Each rule node corresponds to the antecedent α_i of a fuzzy rule *i*, calculating their degrees of fulfilment.

d. **Layer 4: Output Terms**
The overall support sup_j for each fuzzy output term *j* is calculated by integrating all antecedents α_i concerning these output terms from the previous layer applying:

$$sup_j = min\{\sum \alpha, 1\} \tag{68}$$

e. **Layer 5: Linguistic Output**
The defuzzyfication is performed by applying the centre-of-area method. The output terms membership functions are limited to gaussians. Therefore the output y_j can be expressed by:

$$y_j = \frac{\sum_i \mu_{ij}\sigma_{ij}sup_i}{\sum_i \sigma_{ij}sup_i} \text{ (see (88) to (91) for a closer discussion of this equation)} \tag{69}$$

where (μ_{ij}, σ_{ij}) are the parameters of the gaussian output membership functions (see (73)) and sup_j is the support from rule *i*.

(μ_{ij}, σ_{ij}) are also interpreted as the weights between layer four and five.

Training

An off-line training procedure is suggested for NNFCDS, consisting of two phases. First an unsupervised clustering of the (fixed) input data is performed in order to extract the underlying fuzzy rules, resulting in a pre-structured network. Applying a well defined training set, this pre-structuring eliminates most of the potential connections from and to layer three. See also [Kosko92] for another example of extracting fuzzy rules from fixed data sets. The second training phase contains a gradient decent approximation of the overall neuro-fuzzy system to the training set. This is performed in a supervised manner, where heuristics have to be applied, overcoming the restrictions of the non-differentiable fuzzy-implication respectively defuzzyfication implementations. The applied "gradient-heuristics" can be found in [LinC91].

In the following, the rule extraction procedure as proposed from Lin and Lee will be discussed briefly. The data set analysis is divided into four stages: initialization, clustering, rule extraction, and pruning

a. **Initialization**
The number of input respectively output terms has to be chosen in advance, and determines the whole connective structure of the network, beside the number of nodes and connections in layer three (rule layer). Here any possible combination of input terms is considered as a potential connection from layer two to layer three and thus installed. Finally, layer three has to be fully connected to layer four.

b. **Clustering**
A self-organizing map algorithm is applied, clustering the input data (the modification refers to the neighbourhood adaptation). It is assumed that the initial choose of the gaussian centres μ_{ij} is well defined, because the modified Kohonen algorithm would not converge to the intended input terms otherwise. The results of the first clustering step are the stabilized centres of the gaussians of the input terms. The width of the input gaussians can then be approximated by:

$$\sigma_i = \frac{\| \mu_i - \mu_{closest} \|}{r} \tag{70}$$

where r determines the degree of overlapping between the gaussians.

c. **Rule Extraction**
A competitive learning algorithm is employed for the determination of the weights between layer three and layer four. The proof for convergence regarding this competitive process can be found in [LinC91]. Among the different terms out of each term set attached to the linguistic outputs, only one can be activated by each rule node. This (reasonable) limitation in the consequences of the fuzzy rules, limits the number of extracted connections additionally. The time complexity (assuming SISD computers) for this process is:

$$O\left(\prod_{i=1}^{n} |T(X_i)| \cdot \prod_{i=1}^{m} |T(Y_i)| \cdot |L| \right) \tag{71}$$

where n is the number of linguistic variables X; m is the number of linguistic output variables Y; and L is the applied learning set.

d. **Pruning**
Pruning is performed by detecting "overlapping" rules, i.e. rule combinations, replacing (and substituting) multiple rules generated in step c are forced. Redundant rule sub-sets can be detected according to the following criteria:

d-1 The consequences of all rules out of the rule sub-set are identical, and

d-2 some input terms are common to these rules, and

d-3 for all input term combinations, not included in the common input terms, a rule can be found having this combination (and of course the common input terms) as its input[6].

This process might result in a further reduction of rule nodes.

Discussion

The introduced NNFCDS model by Lin and Lee offers high flexibility and supports even the pre-structuring process of the system, where the amount of expert knowledge, is reduced to the determination of the

6. Due to a fault in the original article, this criterion is corrected to the presented form in [Bruske93b].

number of linguistic input and output terms. Nevertheless, two major weaknesses can be found in this model. First the computational effort (and even the space complexity) regarding the pre-structuring process is rather high, assuming SISD computers. This could be overcome, applying parallel computers, optimized for such typical neural processing, like competitive learning, and self-organizing maps. The second point is that the "gradient-heuristics" needed for the second learning phase can fail totally. Thus the *SPIN-NFDS* system, introduced in the next section is built on the base of differentiable functions in all layers.

9.3. SPIN-NFDS – A Neural Fuzzy Decision System

According to the general model of a **fuzzy-decision-system** (**fds**), all the needed components (figure 16) are discussed first in order to define a mapping on a neural net structure in the following. Finally some short remarks about the learning techniques will be made.

9.3.1. The Structure of the Fuzzy Decision System

Rule-base

The rule-base in *SPIN-NFDS* consists of linguistic rules, which obey the following syntax:

<lrule> ::= if <antec> then <conseq> [else <lrule>]

<antec> ::= <propos> [and <antec>] | anything

<conseq>::= <propos> [and <antec>]

where <propos>::= <variable name> is <term name>

where <term name> denotes a linguistic value of the linguistic variable <variable name>. The "anything" antecedent is used to define consequences as unconditioned "facts" in the rule-base. The actual rule set is translated to fuzzy relations according to definition 11, 12, and 18.

Membership functions

Membership functions are used according to the definition of linguistic variables, in order to define the linguistic values of the associated term set. The membership functions are restricted in this system to the following basic types.

- Sigmoid function:

$$S(\mu, \sigma, x) = 1 / \left(1 + e^{-\frac{x-\mu}{\sigma}}\right) \tag{72}$$

  In the following, a sigmoid function with $\sigma > 0$ is called a **rsigmoid function** and with $\sigma < 0$ a **lsigmoid function**.

- Gaussian or radial basis function (rbf):

$$N(\mu, \sigma, x) = e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{73}$$

Both functions may be used for membership functions of terms appearing in an antecedent of a lrule, while only rbf-s may be used in the definition of terms appearing in a consequence.

Fuzzyfication interface

Non-fuzzy input data are mapped to fuzzy sets by treating them as "scaled" fuzzy singletons: If $u_0 \in U$ is a non-fuzzy input data and X is the corresponding linguistic variable with $dom\ (X) = U$ and $T(X) = \{T_x^1, \ldots, T_x^k\}$, the input is fuzzyfied by setting

$$M_{X_0}^i(u) = \begin{cases} M_X^i(u) & ; u = u_0 \\ 0 & ; \text{otherwise} \end{cases} \quad (1 \le i \le k) \tag{74}$$

Inference engine

The fuzzy implication ("then") along with the operation "and" and the combination of several fuzzy rules ("else") has to be defined.

In the first step the evaluation of the "and" in the antecedents is done by a cartesian product.

$$A \wedge B = \{[(u, v), M_A(u) \cdot M_B(v)] \mid u \in U, v \in V\} \tag{75}$$

Since all input terms are scaled fuzzy singletons, equation (75) simplifies to:

$$A \wedge B = \{[(u_0, v_0), \alpha] \mid u_0 \in U, v_0 \in V\} \tag{76}$$

$$\alpha = M_A(u_0) \cdot M_B(v_0) \tag{77}$$

with α is said to be the "degree of fulfilment" for this antecedent. The fuzzy implication is also implemented by a cartesian product, with the membership-function of the antecedents A being represented by a scalar $\alpha(u)$ (because of the scaled fuzzy singletons as the input data).

$$A \rightarrow B = \{[(u, v), \alpha(u) \cdot M_B(v)] \mid u \in U, v \in V\} \tag{78}$$

The combination of multiple fuzzy implications $A_i \rightarrow B_i$ $(1 \le i \le r)$ is interpreted as a union of the resulting fuzzy sets.

$$\{A_i \rightarrow B_i\} = \cup (A_i \rightarrow B_i);\ (1 \le i \le r) \tag{79}$$

$$= \{[(u, v), max_{1 \le i \le r} \{M_{A_i \rightarrow B_i}(u,v)\}] \mid u \in U, v \in V\} \tag{80}$$

Since differentiable functions are needed for the learning techniques in the later neural network section, the *max*-function is approximated by *sum* as defined below.

$$sum: [0, 1]^r \rightarrow [0, 1]; \tag{81}$$

$$sum_{1 \le i \le r} (M_{C_i}(w)) = \frac{\sum_{i=1}^{r} M_{C_i}(w)}{\sum_{i=1}^{r} max_{w \in W} \{M_{C_i}(w)\}} \tag{82}$$

Of course the definition itself contains a *max*-function again, but this one reduces to a linear function, as can be seen in the following example of a complete rule evaluation.

Let "**if** $X_1 = A_i$ **and** $X_2 = B_i$ **then** $Y = C_i$" be a rule base of r rules $(1 \le i \le r)$. In a first step the antecedents reduce to

$$\alpha_i = M_{A_i}(x_0) \cdot M_{B_i}(y_0);\ (1 \le i \le r) \tag{83}$$

and the output terms can be calculated to:

$$C'_i = \{[w, \alpha_i \cdot M_{C_i}(w)] \mid w \in W\} \tag{84}$$

The membership function of the union of the output terms, using the *sum*-equation (81), may be written as

$$M_Y(w) = \frac{\sum_{i=1}^{r} \alpha_i \cdot M_{C_i}(w)}{const \cdot \sum_{i=1}^{r} \alpha_i} \tag{85}$$

Defuzzyfication interface

Let Y be one of the output fuzzy sets from the inference step (86). Then the centroid method (88) is employed for the defuzzyfication to the non-fuzzy output data y.

$$Y = \{[w, M_Y(w)], w \in W\} \text{ with} \tag{86}$$

$$M_Y(\mathrm{w}) = sum_{1 \le i \le r} (\alpha_i \cdot M_{C_i}(w)) \tag{87}$$

Note that only output terms C_i from the term set of the linguistic variable associated with Y have to be taken into account.

$$y = \frac{\int w \cdot M_Y(w)\, dw}{\int M_Y(w)\, dw} \tag{88}$$

This technique is also known as "centre of area method" (coa) or "centre of gravity".

$$y = \frac{\sum_{i=1}^{r} \alpha_i \int w \cdot M_{C_i}(w)\, dw}{\sum_{i=1}^{r} \alpha_i \int M_{C_i}(w)\, dw} \tag{89}$$

Since the system is restricted to gaussian functions in the output terms, (89) becomes

$$y = \frac{\sum_{i=1}^{r} \alpha_i \int w \cdot e^{-\frac{(w-\mu_i)^2}{2\sigma_i^2}} \, dw}{\sum_{i=1}^{r} \alpha_i \int e^{-\frac{(w-\mu_i)^2}{2\sigma_i^2}} \, dw} \tag{90}$$

and after some simplifications

$$y = \frac{\sum_{i=1}^{r} \alpha_i \mu_i \sigma_i}{\sum_{i=1}^{r} \alpha_i \sigma_i} \tag{91}$$

9.3.2. Neural Net Architecture

The five layered feed-forward neural network used for *SPIN-NFDS* is outlined in figure 18. The several layers are classified in the sense of the NFD-scheme introduced above and by the calculations being done at each layer. Note that this network architecture is *not* fully connected. A description of the computations at each stage is given in the following. The resulting set of parameters, that may be trained contains the parameters σ, μ of each membership function from the input and the output variables and a set of weights w, one for each output term in the linguistic rules.

Layer 1: Linguistic input

The operation being done here is only transmitting the input values to the terms of the corresponding (i.e. not to all) input variables at the next layer.

Layer 2: Input terms

Let n be the number of linguistic input variables with names X_i $(1 \le i \le n)$, values $(x_1, \ldots, x_n)$, and term sets $T(X_i)$. This layer computes the membership functions

$$M_{X_i}^{j_i}(x_i) \; ; 1 \le i \le n; \; (1 \le j_i \le |T(X_i)|) \tag{92}$$

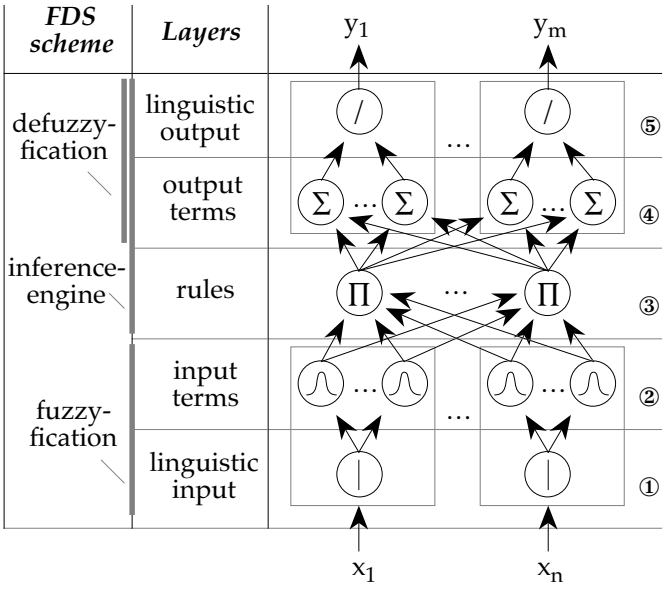

Figure 18: SPIN-NFDS network topology

where sigmoid (72) and gaussian functions (73) are allowed. The parameters in each node, which have to be optimized by learning (described in the next section) are

$$(\mu_i^{j_i}, \sigma_i^{j_i}) \tag{93}$$

Layer 3: Rules

From the membership functions the degree of fulfilment α_i $(1 \le i \le r)$ for each antecedent has to be calculated (with r is the number of linguistic rules in the rule base). The α_is are computed by $\prod$-neurons[7] without any additional weight in the input terms.

Layer 4: Output terms

Let $t = |T(Y_j)|$ and n_i be the number of linguistic rules supporting the i-th term of Y_j. So α_k^i is the support of the k-th rule for the i-th term of Y_j and equation (91) may be formulated as

$$y_j = \frac{\sum_{i=1}^{t} (\mu_i \sigma_i \sum_{k=1}^{n_i} \alpha_k^i)}{\sum_{i=1}^{t} (\sigma_i \sum_{k=1}^{n_i} \alpha_k^i)} \tag{94}$$

The computational step being performed in this layer is the summation of the support for each term from the term sets of the linguistic output variables, i.e.

$$s_i = \sum_{k=1}^{n_i} w_{ki} \alpha_k^i \tag{95}$$

where weights w_{ki} are introduced as a possibility to determine the "importance" of a single rule for an output term. All these weights are initialized to 1 and may be altered during the learning phase.

Layer 5: Linguistic output

Once the supports s_i have been calculated by the previous layer, equation (91) simplifies to

$$y_j = \frac{\sum_{i=1}^{t} \mu_i \sigma_i s_i}{\sum_{i=1}^{t} \sigma_i s_i} \tag{96}$$

Note that there is a certain redundancy concerning the parameters σ_i and w_{ki}.

9.3.3. Learning in SPIN-NFDS

Training is performed by standard backpropagation techniques. Two schemes are supported: The "usual" backpropagation (also called: off-line, batch, epoch) and the "on-line" backpropagation (also referred to as sample, or stochastic backpropagation).

The applied network error, which all the corrections are based on, is defined as the common square error E_T with respect to a training set T:

$$E_T = \frac{1}{2} \sum_{(x^T, y^T) \in T} (y^T - NFDS(x^T))^2 = \frac{1}{2} \sum_{(x^T, y^T) \in T} \left(\sum_{j=1}^{m} (y_j^T - y_j^{NFDS}(x^T)) \right) \tag{97}$$

where $y_j^{NFDS}(x^T)$ denotes the j-th output term of the *SPIN-NFDS*-system applying x^T.

This network error E_T is used in case of batch-error-backpropagation, where all the elements in the test set T are evaluated in order to get a good base for the following gradient decent method. If a "on-line" correction is need, the error term $E_{(x,y)}$ can be calculated for each training pair individually:

$$E_{(x,y)} = \frac{1}{2} \sum_{j=1}^{m} (y_j - y_j^{NFDS}(x)) \tag{98}$$

"On-line" learning according to $E_{(x,y)}$ results in worse approximations for the gradients (in comparison to batch-error-backpropagation), but limiting the learning constants to small steps producing a smooth behaviour nevertheless. Moreover the stochastic element included in the spontaneous learning technique can avoid local minima in some situations.

7. A $\prod$-neuron calculates its output to $out_{\prod} = \prod x_i$ with $x_1, \ldots, x_n$ being the inputs of the neuron.

Based on E_T or $E_{(x,y)}$ (denoted with E in the following) a gradient decent on the network weights is implemented. For the last layer n in the network this results in:

$$w_{ij}^{n,new} = w_{ij}^{n,old} + \eta_{ij}^{n} \frac{\partial E}{\partial w_{ij}^{n}} \tag{99}$$

where η_{ij}^{n} is the learning constant for the weight w_{ij}^{n}.

In all previous layers the changes of the weights are calculated recursively according to:

$$w_{ij}^{l,new} = w_{ij}^{l,old} + \eta_{ij}^{l} \delta_j^l \frac{\partial net_j^l}{\partial w_{ij}^{l}} \tag{100}$$

and

$$\delta_j^l = \sum_k \delta_k^{l+1} \frac{\partial net_k^{l+1}}{\partial o_j^l} \frac{\partial o_j^l}{\partial net_j^l} \text{ (the error signal for the neuron } j \text{ in layer } l) \tag{101}$$

where o_j^l is the output and net_j^l the network input of neuron j in layer l.

The actual gradients for the *SPIN-NFDS*-system are calculated and shown in [Bruske93b] pp.46-49. A wide range of optimizations (especially speed-ups) can be applied to a backpropagation system (see [Hertz91] for a set of common optimizations). In the context of the *SPIN-NFDS*-system a momentum term (102) has been introduced, but due to the already very high learning rates as discussed in the next section, the actual speed-up is not essential. On the other hand, a momentum term (as most of other potential speed-ups) can be a significant risk regarding the stability and convergence of the whole system.

$$w_{ij}^{l,new} = w_{ij}^{l,old} + \eta_{ij}^{l} \delta_j^l \frac{\partial net_j^l}{\partial w_{ij}^{l}} + \alpha \Delta w_{ij}^{l,old} \tag{102}$$

where $\Delta w_{ij}^{l,old}$ denotes the previous change in $w_{ij}^{l,old}$.

As usual, the learning constants are treated in groups corresponding to the layers of the network.

9.3.4. Simulations

Several tests have been done, according to the following test strategies for *SPIN-NFDS*:

- Analysing the *training performance*, with an "ad-hoc" rule base for our decision task.
- Analysing the *generalisation performance*, i.e. training and testing is being done with different sets of test surface-fragments.
- Comparing performances resulting from changes in the size of the learning set.
- Comparing performances resulting from changes in the predefined rule-base.
- Comparing performances from *SPIN-NFDS* to that of a "classic" backpropagation classifier.

The employed rule base is intentionally simple and far from an "optimal" solution. Elementary geometric considerations (introduced in the motivation of this chapter) like smooth continuations are used to formulate the rules. The found rule base gives rather worse results, regarding the decision problem. Due to the general idea that global geometric rules will not sufficiently approximate the specific surface fusion problems given by different environments, it has not being tried to optimize the rule base beyond certain limits.

Some of the most interesting results are shown in the following. For the complete discussion see [Bruske93b]. The training sets T_1 and T_2 applied in all tests, consist of 300 examples of fragment-pairs each.

Iterations	0	1	5	10	50	175
Class. rate[%] $T_1^{0\%}$	50	50	96	96	97	99
Class. rate[%] $T_1^{10\%}$	50	50	50	63	92	98

Table 2 : Training performance with 0% and 10% noise

Training performance

As the reader may see from table 2, the classification rate rises very fast, considering a backpropagation system, even for a less optimized rule base. The initial classification rate for the rule base applied here is only 50%. This fast convergence is slowed down by adding noise, but nevertheless a classification rate beyond 90 % is reached with 50 training steps.

Classification rates [%] after 70 iterations		trained on $T_1^{0\%}$	$T_1^{5\%}$	$T_1^{10\%}$
classifies	$T_2^{0\%}$	98	98	98
	$T_2^{5\%}$	83	96	97
	$T_2^{10\%}$	63	85	95

Table 3 : Generalization performance

Generalisation performance

Interesting, but not completely surprising are the results of the tests for generalization. The reader may see from table 3 that the training on crisp training sets shows only poor results in noisy environments, whereas training on noisy training sets shows reasonable results on noisy and crisp test sets.

Chapter 10 Surface Classification

Based on the surface representation described in section 7.3, the continuous flow of scanned and pre-processed surfaces is to be clustered respectively classified based on a norm applied to the surface-vectors.

10.1. Network Model

The part of *SPIN* discussed in this chapter requires a network model, with features listed below:

- Unsupervised clustering
- Dynamical number of clusters
- Forgetting controlled by time or frequency of access
- Flexibility over an infinite period of time

There is only a small number of well known networks that might be used for such purposes (e.g. ART-models by Carpenter & Grossberg [Carpenter91a], [Carpenter91b], [Carpenter88], [Carpenter87a], [Carpenter87b], [Grossberg87c], [Moore88], Self-organizing feature-maps by Kohonen [Kohonen90], [Kohonen84], GAL by Alpaydin [Alpaydin91]). But all show limited abilities in at least one of the mentioned aspects. None of the methods is prepared to handle flexibility over an infinite period of time. Variable number of classes is not supported by the Kohonen-structure. Finally, controlled forgetting is not considered by Kohonen and Alpaydin.

10.1.1. Dynamic Network Model

The base of the following network-model is the self-organizing feature-map model by Kohonen [Kohonen90]. The well known structure is extended by the possibility of adding and removing new cells. This work was done by Fritzke in 1991 [Fritzke91a], [Fritzke91c]. Although being published already, the main aspects of this extension will be shown in short form before discussing further adaptations.

Generalization & Learning

The representation of the surfaces as shown above is a vector in a m-dimensional real-valued vector-space. These vectors are mapped to an array S of cells c, each attached to an m-dimensional position vector $pos(c)$. The cells in S are connected in a simplex structure (the original Kohonen-model uses a rectangular structure). For clearness but without loss of generality, triangles will be used as the simplex structure in all following discussions. An input vector x is then mapped to the cell with the smallest distance to x (in the Eu-

clidian norm). This cell is called *bmu* (best matching unit) and this procedure is referred to as the classification.

$$\forall c \in S: \| pos(bmu) - x \| \leq \| pos(c) - x \| \tag{103}$$

For learning purposes the cell *bmu* and its topological neighbours in the triangular structure are "moved" towards the input vector (by a fraction e_{bmu} and $e_{neighbour}$ of the distance). This training step is repeated n_d-times.

The described procedure results (if well defined) in a *topology-preserving* (i.e. adjacent input vectors are mapped on adjacent cells) and *distribution-preserving* map (i.e. the density of the cells approximates the probability density *P(X)* of the input vectors).

Growing cell-structures

Up to here, the number of cells in *S* is constant, i.e. the number of cells that should represent the probability-distribution of the input vectors adequately has to be known in advance. Additionally the initial position of the cells is critical regarding the ability and speed of learning. To overcome this restriction a mechanism of expanding the structure is introduced. The initial structure is a single triangle, with arbitrary values of the three cells. In each learning step the distance between the input vector and the *bmu* is added to an error-variable associated with the found *bmu*. After n_d learning steps the cell with the maximal error-variable is detected. This cell is called *bs* (black sheep). Then the farthest direct neighbour in the structure is detected and called *f*. The new cell is inserted in the middle between them:

$$pos(c_{new}) = \frac{1}{2} \cdot (pos(bs) + pos(f)) \tag{104}$$

This new cell must be connected with surrounding cells in order to keep the triangular structure. The error-variable is initialized as a mean value of the error-variables of all d direct connected neighbours:

$$err(c_{new}) = \frac{1}{d+1} \sum_{i=1}^{d} err(neighbor_i) \tag{105}$$

The error-variables of the neighbours are reduced according to that amount:

$$\forall i = 1, \ldots, d: err(neighbor_i) = \frac{d}{d+1} err(neighbor_i) \tag{106}$$

Shrinking cell structures

Starting from a single triangle and expanding the structure as described above will produce a connected structure approximating the probability density *P(X)*. This might be an inadequate model because *P(X)* could consist of several distinct regions with *P(X)*=0 between them. So there is a need for a procedure splitting the different parts of the structure if necessary.

The basic idea is finding cells which are positioned in areas with *P(X)*=0 and remove them. As the indicator of this constellation, the number of classifications *k* without being *bmu* are recorded for each cell. If *k* exceeds the value in (107) for a certain cell, this cell is removed and so are those of its neighbours necessary to return to a structure of triangles (p_s means the probability of keeping needed cells and n is the current number of cells)

$$k > n \cdot n_d \cdot (1 - (1 - p_s^{1/n})^{1/(n_d+1)}) = k_r \tag{107}$$

For a more detailed discussion of the algorithms so far see [Fritzke91a], [Fritzke91c], or [Fritzke93c].

10.1.2. Extensions & Adaptations

A number of problems have being found employing the above network-structure for surface-clustering and classification. The central aspect here is the fact that the applied learning set is not fixed but consists of a continual flow of examples. So the task is not modelling the best approximation of a limited set of input vectors, but finding a representation of the probability distribution and a adequate clustering of the *most recently* presented surfaces. Additionally the classification-function has to be accessible *continuously* (after a certain amount of learned surfaces), i.e. the structure should change states smoothly while learning.

Buffering the flow of examples

The number of produced examples per time-interval is smaller than the number of surfaces that might be used for adapting the model during this interval. So the received examples are buffered in a FIFO-list and each example is applied multiple times.

Limited growing and adaptation

In the original model the learning phase simply stops when the required accuracy is reached. The *SPIN*-project does not know anything about an end of learning, so there must be another definition of stability. Assuming a tolerated error of $d_{accuracy}$ and an input vector x. If the *bmu* resulting from the classification of the vector x fulfils equation (108)

$$d_{bmu} = \| x - pos(bmu) \| \leq d_{accuracy} \tag{108}$$

then the learning for this cell is switched to a modified learning scheme: The fraction of *bmu* correction e_{bmu} is reduced by the factor Δe_{bmu} and the *bmu* is now moved towards x by this reduced fraction of $\Delta e_{bmu} \cdot e_{bmu}$. The direct neighbours are not corrected at all. Additionally this classification does not increment n_d, i.e. the generation of new cells is delayed respectively stopped.

If these classifications continue over some period of time, the moving of cells will become slower and no new cells will be created. The modified learning scheme is reset immediately to normal learning when one classification does not fulfil (108), i.e. e_{bmu} is set to the original value, etc.

Summarizing this modification, two main effects are important:

a. The growing of the network is stopped (or at least slowed down, depending on the classification error), although the flow of examples may still continue.

b. The network structure (i.e. the positions of the cells) is stabilized, when the examples fit into the clustering built up.

c. The network is switched spontaneously to maximal learning flexibility if necessary.

Cautious removal

The removal of cells in the original algorithms causes the removal of neighbouring cells, in order to return to a structure of triangles. This procedure does not care about the importance of these neighbouring cells, so important (i.e. often used) cells might disappear. If the relative density of cells is large, then the remaining cells might fulfil most of the further oncoming classifications. But in a sparse clustering, i.e. each cell represents a whole isolated class (e.g. with $P(X)=0$ at its borders), the functionality of these cells can not be adequately approximated by the remaining ones. And even worse, these cells will be created once again in the further process, because the classification-errors in this area will rise significantly. On the way to this rebuilt structure, the remaining cells are corrected in large steps because of large classification-errors.

In some constellations a cyclic behaviour can be detected, i.e. even in the moment when the structure was built up again, one of the cells in this area is removed, and the procedure starts again. This unstable and discontinuous behaviour can not be tolerated in the present system.

The chosen solution is based on the idea of accepting "over-classification" under some circumstances, but never deleting employed cells. Thus each time a cell is detected as not being used for a certain period of time, it is only removed when all the cells which would be removed in order to keep the structure of triangles intact are also detected as unused. As a result of this manipulation, the network will consist of more cells than necessary, and effects like swapping between two cells in situations where one cell would be sufficient or other instability problems might be considered. None of these potential problems were observed during the simulations.

Speed of forgetting

In the original model the value of k_r might be approximated as (see equation (107)):

$$k_r \leq \lceil n \cdot n_a \rceil ; (0 \leq p_s \leq 1) \tag{109}$$

A value of p_s near 1 implies a good approximation of the vectors in the current learning set. But due to the fact that the learning set is dynamic, another aspect arises: What happens to a learned cell, when the generating examples are deleted from the learning set? In the simulations this cell is removed or even massively corrected in a couple of minutes, i.e. there is absolutely no long-term memory.

In order to create a possibility to determine slower forgetting then implied by p_s=1, the definition is extended of k_r for values of p_s>1 (Notice that p_s is no longer a probability in this case):

$$k_r = \begin{cases} \lceil n \cdot n_d \cdot (1 - (1 - p_s^{1/n})^{1/(n_d+1)}) \rceil & ; p_s \le 1 \\ \lceil n \cdot n_d \cdot p_s \rceil & ; p_s > 1 \end{cases} \quad (110)$$

So arbitrary long storage-times of presented surfaces can be forced. Additional information could be stored in every cell in *S*, like frequency of access or time since last access, etc. pp., in order to find a better choice for cells to delete, but this ideas are not tested in the context of surface classification. For an example of another forgetting strategy ("local forgetting") please refer to section 4.2.3.

10.1.3. Parameters

This section is intended to give an idea of the meaning of some parameters as well as of useful ranges in this context.

Network size

The size of the network structure is not determined directly by an upper or lower bound in the current version, but is implied by the required accuracy of the classification. Therefore this parameter needs not to be tuned.

Accuracy of classification

The accuracy of classification is defined as the euclidian distance between the example-vector and the found *bmu* (see equation (108)). So the first stage of learning is reached when all the vectors of the learning set are in between of at least one of the hyper-spheres with radius $d_{accuracy}$ around the cells of the network. Then the modified learning-phase is entered, i.e. no new cells are created and the speed of moving as well as the number of cells moved in each step are reduced (until the classification errors rises again). The network is called "stable", when e_{bmu} drops below a certain limit e_{stable}.

Choosing $d_{accuracy}$ too small results in a one-to-one mapping of the current learning set and the cells in the structure, i.e. one cell is created for each vector in the learning set. On the other hand a large value of $d_{accuracy}$ means a small number of created cells and a large tolerated classification-error. Finding an "optimal" value depends widely on the purpose of the system and the employed vector-representation.

Moving fractions e_{bmu}, $e_{neighbour}$, and Δe_{bmu}

Simulations have shown good results with values in range given by equation (111) for e_{bmu}.

$$\frac{1}{20} \le e_{bmu} \le \frac{1}{5} \quad (111)$$

A "working range" for $e_{neighbour}$ is shown in (112), but some further restrictions must be considered.

$$\frac{1}{50} e_{bmu} \le e_{neighbour} \le \frac{1}{20} e_{bmu} \quad (112)$$

A small value for $e_{neighbour}$ might result in a not topology preserving cell moving in the network-structure, because one cell, which is marked as the current *bmu* could be moved over a long distance, without moving the surrounding cells in an adequate manner. Large values for $e_{neighbour}$ are dangerous too, due to the fact that if a neighbouring cell of a critical-cell (i.e. not yet well adapted) becomes the *bmu*, the correction distance of this critical-cell could be larger than in the case that the critical-cell itself is the *bmu*.

Δe_{bmu} is limited due to the following discussion. When the value of e_{bmu} remains below the limit e_{stable}, the network is said to be stable. But this should imply that every vector of the learning set is classified *at least once* with a classification error smaller than $d_{accuracy}$:

$$e_{bmu} \cdot \Delta e_{bmu}^{|Learning\ set|} > e_{stable} \quad (113)$$

or

$$\left(\frac{e_{stable}}{e_{bmu}}\right)^{1/|Learning\ set|} < \Delta e_{bmu} < 1 \quad (114)$$

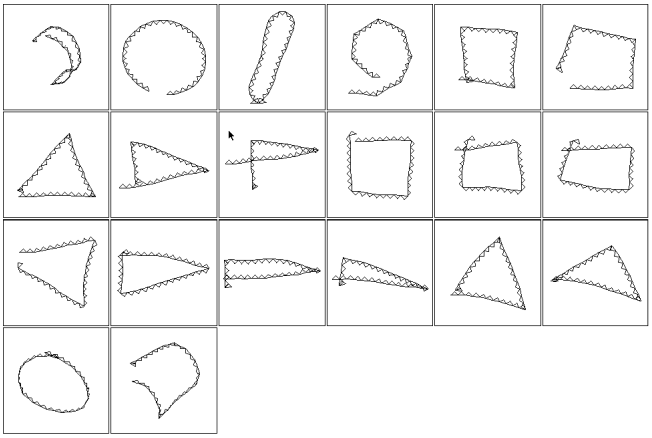

Figure 19: Some of the input surfaces

where $|\,Learning\ set\,|$ is the number of example vectors in the learning set

These are the obvious limits only. Both bounds would not result in a reasonable learning behaviour. A value of Δe_{bmu} too near to 1 generates a very slow convergence of the network without reaching a better accuracy, and a value too near to the lower bound results in a system resting too early i.e. with sub-optimal positions of the cells. Simulations have shown good results for values in the range of:

$$\Delta e_{bmu} = \left(\frac{e_{stable}}{e_{bmu}}\right)^{\frac{1}{c|Learning\ set|}}; c \in [3{,}6] \tag{115}$$

Removal of cells

An uncritical behaviour (with the new learning scheme) could be shown in simulations with values of p_s larger than 0.5. Further increasing of this value results in an increased time a learned example vector (or class of vectors) remains in the network, without the need to "refresh" the corresponding cell with this example vector(s).

Creation of new cells

The factor n_d, which determines the number of learning steps before a new cell is inserted, should be chosen in a way that "just enough" probability density information is accumulated before a position for a new cell

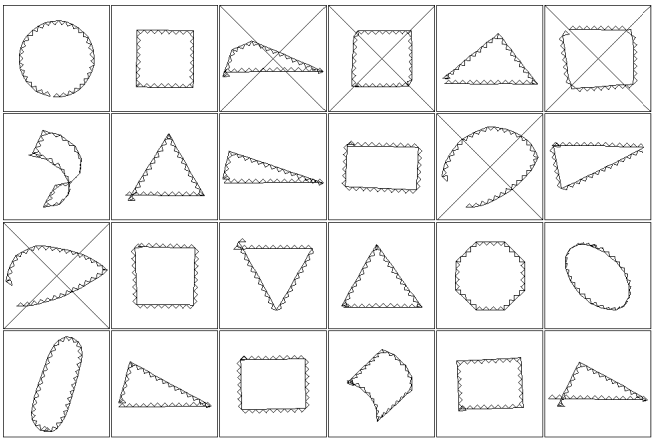

Figure 20: Generated classes

is determined. Twice the number of expected classes works as a rough approximation, but obviously this "just enough" depends widely on the actual distribution of the example vectors.

10.2. Simulations

A flow of examples consisting of 20 different types of surfaces, which are all quantised to 32 values for the segment curvature and in 16 entries for the surface curvature was simulated. The number of generated different examples is 2000. Noise is added in the form that every orientation of the bordering surface pieces is disturbed in the range of ± 6° for the segment orientation and ± 9° for the surface orientation. Some of the generated examples are shown in figure 19. As a result of adding noise some surfaces are drawn as "not closed", but this does not influence the generality of the test-set. After 6500 learning steps (110 seconds on a MC68030 processor with a 40 MHz clock) 19 different classes are generated ($n_d = 40$, $e_{bmu} = 0.1$, $e_{neighbour} = 0.005$, $\Delta e_{bmu} = 0.998$). In figure 20 the surface-classes are shown with unused (i.e. employed for keeping the simplex structure consistent only) cells crossed out. The shown example is representative, in the sense that all of the well defined test-sets have caused such a network behaviour in the simulations.

Chapter 11 Surface-Cluster Classification

The final processing stage of the *SPIN*-project is described in this section. The central structure handled here are groups of connected 3 -d freeform surfaces, extracted and completed from the surface model. These groups of surfaces are called "**surface-clusters**". Due to the fact that the applied grouping strategies are adapted by the flow of typical structures from the environment, i.e. not by expectations in a predefined model base, the term "object" is avoided, in order not to associate objects, where any kind of (non-geometric) meaning is attached.

The task discussed here is a continuously adapted definition of representative classes of surface-clusters. The only criterion for the definition of a specific surface-cluster-class is the fact that a group of surface-clusters is statistically representative in the current universe of data going throw the processing pipeline. Based on this stable environment-representation, a following (symbolic) object-recognition system should be able to construct and recognize objects, which are meaningful for a certain application.

The implementation of the two ART 2 network models (described in the next section) were done by Keuchel [Keuchel94], whereas the training sets and graphical representations are generated by Wagner [WagnerM93].

11.1. Adaptive Resonance Theory (ART 2, ART 2-A)

The author likes to motivate the reasons for the selection of the Adaptive Resonance Theory (ART) for the surface-cluster classification task in this section. The technical details about these well-known techniques introduced by Carpenter and Grossberg can be found in [Carpenter91a], [Carpenter91b], [Carpenter88], [Carpenter87a], [Carpenter87b], [Grossberg87c], [Moore88]. Due to the fact that among the basic ART-models, ART 1 is designed for binary inputs only, and ARTMAP is a supervised version, the two models: ART 2 respectively ART 2-A are remaining to be tested within the *SPIN*-project. An introduction to the ART-philosophy is skipped here completely, because the models are rather complicated and cannot (at least due to the abilities of the author) described in an obvious and abstract way without losing important details. Moreover, very detailed and motivating introductions can be found in the articles mentioned above. Thus a list of features, which have forced the application of ART in the *SPIN*-project are given here only:

- **Competitive clustering**
  An unlimited number of analog inputs are classified according to the training-sets presented before.
- **Normalization and contrast enhancement**
  These two feature are offered in the model itself. Thus these steps can be skipped in the preprocessing phase.

- **Unlimited flexibility**
  Any kind of "cooling down" of learning parameters is unknown to the ART models. The adaptation abilities and speeds are constant during the whole lifetime of the system.
- **Spontaneous learning abilities**
  Large learning performances can be expected, due to the fact that some learning rules contain spontaneous instead of gradient decent or other "smooth" adaptations – a major advantage of ART regarding the adaptive methods discussed up to here.
- **On-line learning**
  The lifetime of the system is not divided into a learning and an execution phase – a very important feature in the *SPIN*-context. Nevertheless, adaptation can be suppressed, if that should be necessary.
- **High adaptation speed**
  The number of learning steps (especially in the ART 2-A model) needed for a satisfying adaptation is very low in comparison to most other clustering techniques. This is another necessary feature for any on-line application.
- **Biological inspiration**
  The physiological motivation, inspirations, and similarities are often mentioned and emphasized in the articles of Carpenter and Grossberg. Due to the limited competence of the author in this field, this motivation is noticed, but does not have any significance for the evaluation of the models.

Some other (in the *SPIN*-context negative) features might be noticed also:

- **Fixed number of classes**
  This limitation refers to the original articles, but there are obvious possibilities to overcome this problem.
- **Critical initialisation**
  Due to the fact, that the whole network structure is build up from scratch on, some competitive parts of the network must have preset (and different) values. This has turned out not to be a serious problem for the actual task here.
- **Complex structure**
  The normalization, contrast-enhancement, and selective attention subsystems are completely implemented employing neural models. Thus some parts seem to be more complex than they would have been applying "conventional" methods. The author has not tried to overcome this weakness, because the learning speed is satisfying. On the other hand, the strict limitation to a few (neural-) operators in the ART-models results in a very "consistent" and "elegant" structure. Of course, these are no principal arguments, but they play a role during the implementation and debugging phase.

The expected learning speeds in the ART 2-A model are one or two magnitudes higher than the speeds from the ART 2 model, because of simplifications in the transfer functions and the competitive layers together with "quick binding" or "spontaneous adaptation" in some constellations. On the other hand the slower adaptation and some other features of ART 2 promise more stability. Therefore both models are discussed in the tests from section 11.3.

11.2. Surface-Cluster Representation

Any network clustering method needs a representation vector of fixed length, in order to apply a standard norm for the distance-measurements occurring at several stages (for example at the competitive layer). The number of surfaces in a surface-cluster is variable. Therefore a description in the form of curvatures or other features in equidistant border points cannot be applied here. Moreover the descriptive vector should be kept small to ensure an effective clustering. A number of general statistical features have been calculated for each surface-cluster. The complete set of features together with example values calculated at a truncated pyramid, where one of the six surfaces was missing is shown in table 4. It has turned out during tests discussed in the next section that 18 out of these 40 features are sufficient for the test set classifications following.

Surfaces							
No. of surfaces	Size	Largest size	Smallest size	Mean size	Largest size (rel.)	Smallest size (rel.)	Mean size (rel.)
5	1.655 m^2	0.604 m^2	0.059 m^2	0.331 m^2	0.365	0.036	0.139

Surface corners				
Maximal number	Minimal number	Mean number	Obtuse corners	Most appearing no
4	4	4	4	5

Surface corner angles				
Mean	Maximum	Minimum	Obtuse	Acute
89.98°	150.02°	59.98°	6	14

Edge angles				
Mean	Maximum	Minimum	Obtuse	Acute
88.56°	149.05°	59.01°	3	5

Edges							
Total number	Total length	Longest edge	Shortest edge	Mean edge length	Longest (rel.)	Shortest (rel.)	Mean edge (rel.)
8	4.460 m	1.260 m	0.140 m	0.558 m	0.125	0.283	0.031

Corners								
Total number	Max. no. of surfaces	Min. no. of surfaces	Mean no. of surfaces	Most likely corner	Mean no. of corners	Obtuse corners	Acute corners	Symmetrical corners
4	3	3	3	4	3	2	2	2

Table 4 : Surface-cluster features

11.3. Simulations

The tests with several sets of surface-clusters are performed with both models (ART 2 and ART 2-A). The focus of the simulations presented here is on the three following aspects:

- **Learning performance (convergence)**
  The number of learning epochs and the actual computation times, until a stable and useful state of the network is reached are measured.
- **Generalization**
  Based on a trained network, a new but similar training set is presented and classified without any further adaptation.
- **Flexibility (incremental learning)**
  The first test for flexibility contains a continuously growing learning set. The second version of this test assigns a spontaneous change to a new learning set consisting of completely new samples.

The model parameters are chosen according to some preceding investigations (discussed in [Keuchel94]) and set to constant values for all following simulations (see table 5 and table 6). Due to the fact, that both

ART models show reasonable behaviours in all tests applying the shown parameters, it is no real limitation to exclude the network parameters from the degrees of freedom from the main simulations.

Parameter	Value	Comments
z_{up}-init	0.5	normalized initial value for weight vectors
b	0.2	learning constant
q	0.02	threshold for signal function
r	0.99	classification accuracy
a	10	---
b	10	---
c	0.1	---
d	0.9	---

Table 5 : Parameters for ART 2

ART 2-A needs fewer parameters, and the first four have similar or identical meaning as the parameters from the ART 2 model.

Parameter	Value	Comments
z_{up}-init	0.5	normalized initial value for weight vectors
b	0.2	learning constant
q	0.02	threshold for signal function
r	0.99	classification accuracy

Table 6 : Parameters for ART 2-A

Three sets of surface-clusters are generated (table 7). All surface clusters are generated by deleting some surfaces from the basic forms (e.g. cubes, prisms, etc.) and scaling the remaining object randomly in all dimensions (with some limitations depending on the basic form). Thus all entries in the sets are unique, but some of them are similar to a certain degree. The same basic forms are employed in the generation of the learning and test set, whereas other forms are applied in the alternative set (used for incremental learning and generalization tests).

Out of the cluster features described in section 11.2, a set of "significant" features is extracted. The idea is that features with a large variance within one class of surface-clusters do not have an important impact for the quality of the class. The following strategy was chosen to divide significant from non-significant features.

(1) Assign all features to the set of significant features.

(2) Apply the features from the set of significant features to the learning-set and cluster the set until a stable classification occurs.

(3) Calculate the range of values (minima and maxima) for each applied feature and for each generated class.

Learning set (46)	Test set (44)	Alternative set (45)
Octahedrons		Prisms
8	7	10
Pyramids		Truncated pyramids
14	14	14
Rectangular solids (rectangular parallelepipedons)		Cubes
14	14	7
Truncated tetrahedrons		Tetrahedrons
10	9	14

Table 7 : Applied learning and test sets

(4) Select features, with a range of values larger than a certain threshold. Features are only included in this selection, if this wide range can be observed in multiple classes.

(5) If there could be some features selected in step 4 then
- exclude these features from the set of significant features; continue with step 2

otherwise
- stop

The remaining (significant) features (18 out of 40) after applying the above procedure are marked grey in table 8.

Surfaces							
No. of surfaces	Size	Largest size	Smallest size	Mean size	Largest size (rel.)	Smallest size (rel.)	Mean size (rel.)
Surface corners							
Maximal number	Minimal number	Mean number	Obtuse corners	Most appearing no			
Surface corner angles							
Mean	Maximum	Minimum	Obtuse	Acute			
Edge angles							
Mean	Maximum	Minimum	Obtuse	Acute			
Edges							
Total number	Total length	Longest edge	Shortest edge	Mean edge length	Longest (rel.)	Shortest (rel.)	Mean edge (rel.)
Corners							

Table 8 : Remaining surface-cluster features

Total number	Max. no. of surfaces	Min. no. of surfaces	Mean no. of surfaces	Most likely corner	Mean no. of corners	Obtuse corners	Acute corners	Symmetrical corners

Table 8 : Remaining surface-cluster features

Learning performance (convergence)

The number of learning epochs, until a stable behaviour can be detected is very small (for example in comparison to any backpropagation system). A clustering is regarded as "stable", when three successive learning set classifications are identical. Thus the number of learning epochs in table 9 is are quite pessimistic. A stable classification occurs often after one or two learning epochs, but the state "stable" is not yet accepted by the observer. The number of learning steps per second is of course a very rough approximation depending widely on the actual implementation. In the test-implementation a number of (graphical) outputs are done during the learning phase. This is one of the explanations for the quite low difference in learning speeds between ART 2 and ART 2-A. Other effects are that steady-state values of the differential equations are employed and the (competitive) F_2-layer of ART 2 is implemented allowing one winner only. The classification results using other training sets or similar parameters are comparable. Depending on the order of presentation, nine to eleven classes are generated, and the classification is changed only very slightly during the adaptation phase, i.e. the assignment of a surface-cluster to a certain class is done only once and hardly ever changed until a stable state is detected. The classification shown in figure 21, produced by the ART 2-A network with sequential presentation is representative for all classifications with the learning set from table 7 employing the parameters from table 5 and table 6. Whether the found classes are really suitable for a following symbolic object recognition system or not, cannot be answered ultimately. On the other hand, the visual impression from figure 21 makes an useful application plausible, when applying "human" criteria for the classification semantic.

ART 2	ART 2-A
Learning epochs (sequential presentation)	
3.5 … 5.5	3 … 4
Learning epochs (stochastically presentation)	
7 … 8	7 … 11
Learning steps / s (68040, 30 MHz) (rough approximation!)	
25	55

Table 9 : Learning performance

Generalization

The classification of the training set (shown in figure 21) is the basis for the generalization test. Without any further adaptations the test set and the alternative set are applied for classification to the same networks. The results of these classifications can been seen in figure 22 for the test set, and in figure 23 for the alternative set. Only two entries could not be assigned to the established classes, whereas the rest of the sets fits very well in the previously learned classification schema. The shown results are produced with the ART 2-A model, although the classifications generated with the ART 2 model are equivalent or slightly better (only one not assigned surface-cluster). The shown generalizations are astonishingly good. Therefore the "counter-test" (i.e. training the model with the alternative set and classifying the remaining sets) was done also. In this case twelve (instead of two in the test above) surface-clusters out of the training set remain unclassified. But the interpretation of this worse generalization is, that the alternative set is not as representative as the training set was for the universe of surface-clusters applied here, i.e. this second test does not say anything about the generalization abilities of the ART models, but shows simply the fact that generalization can only be done on the base of representative inputs.

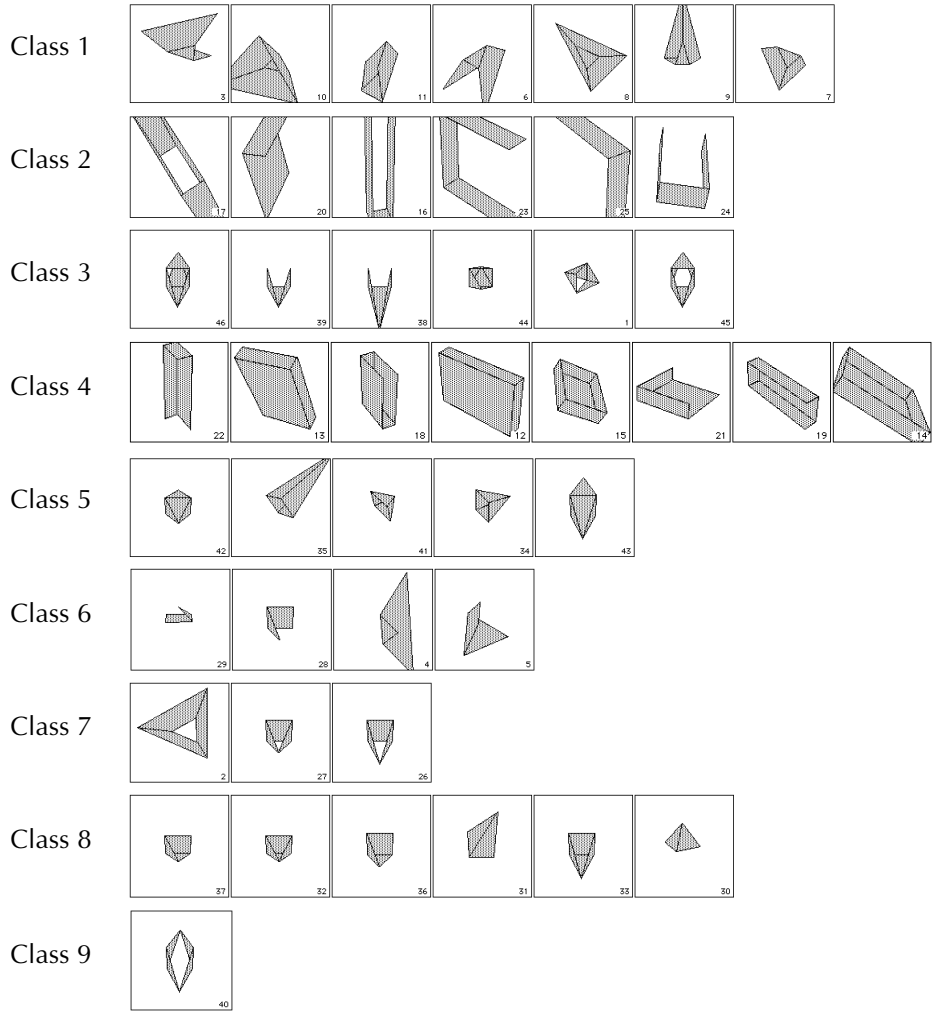

Figure 21: Classification of the learning set with a ART 2-A model

Incremental learning

The tests for incremental learning are processed in two phases. First the learning set is increased dynamically (but still including the former entries) and second two different sets are applied alternating for the adaptation.

The first test-phase (**increasing learning sets**) is based on the stable classification of the training set from figure 21. The training set was then mixed with the alternative set and presented to the ART 2-A model. After four additional learning epochs a stable classification could be established, where two new classed were added. The new classes are the natural consequence of the two unclassified surface-clusters from the generalization test with the alternative set. Due to some minor changes in the classifications, one of the two entries, which have forced new classes, was included in one of the old classes, when the adaptations came to a stable state. Thus only one additional class was needed to include the alternative set, and the number of training epochs reaching this state is still very small. Other slight movements of the classification borders in order to represent both sets, were reasonable. The same test with the ART 2 (instead of the 2-A) model, results in an even more stable behaviour and the same number of additional adaptation steps. None of the classifications from the learning set were revised during the second adaptation. Moreover no additional classes were needed and minor changes in the classification borders were sufficient to fit the complete alternative set into the previous classification schema.

In the second test phase (**alternating learning sets**), only the alternative set was applied to the ART 2-A model previously adapted with the training set (figure 21). The classification results regarding the elements from the alternative set were very similar to the test above, but when the training set is classified (not trained) again, two of the entries from the training set cannot be classified any longer. They have been "washed-out" from the model. Two more training phases (first the learning set, then the alternative set

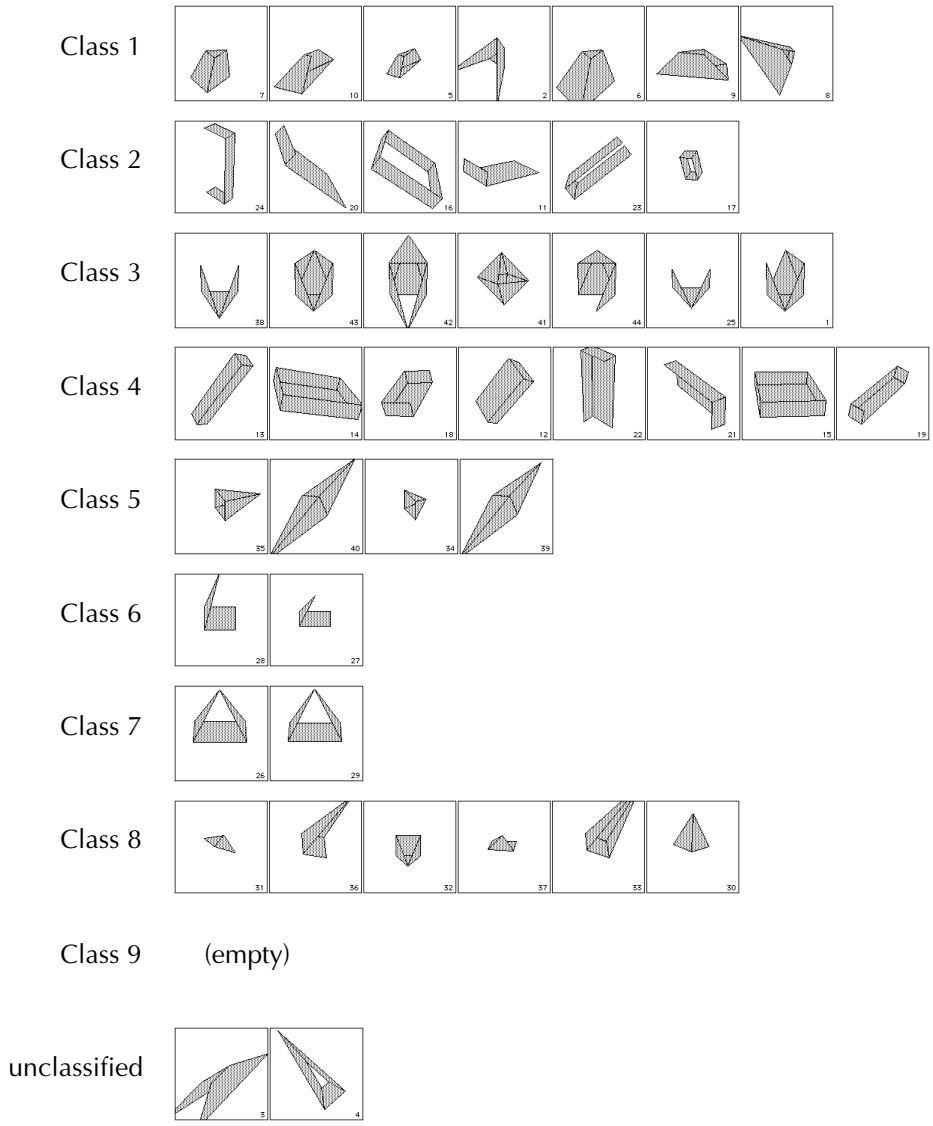

Figure 22: Classification of the test set with a ART 2-A model

trained again) with four training epochs each were needed to get a classification which was satisfying for both sets. Circular adaptations, i.e. the adaptation was swapped between two stages where each of them were not satisfying for one of the training sets, could be detected in some simulations. In any test-case a small number (4-6) of alternating training phases were sufficient for a stable classification, i.e. including both sets completely. Nevertheless this is a very weak argument, because it could not be shown that a convergent behaviour will occur in any case.

The "alternating learning sets"-evaluation with the ART 2 model was significantly better than the test with ART 2-A described above. Slight movements of the classification borders could be detected, but nothing from the learning set was "washed-out" and no cyclic behaviour occurs at all. The slight but reasonable classification movements regarding the learning set are shown in figure 24.

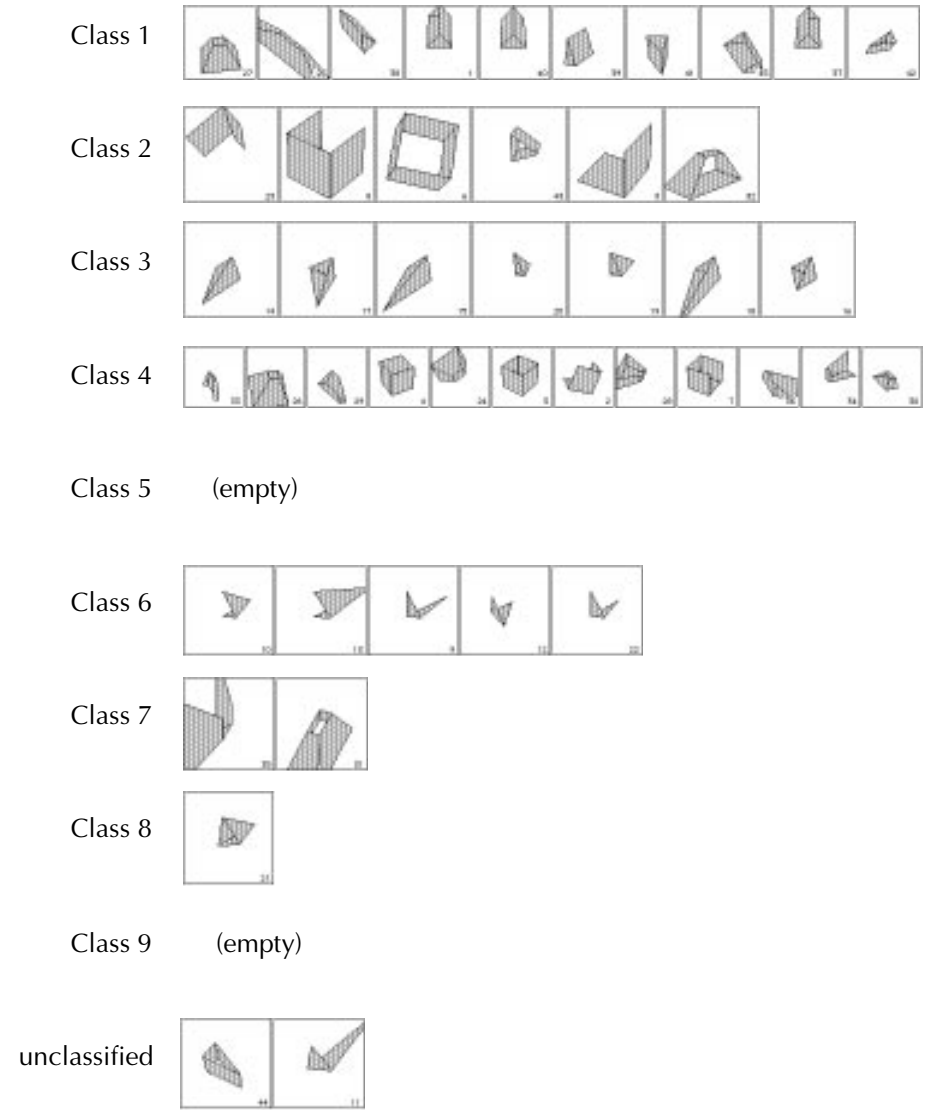

Figure 23: Classification of the alternative set with a ART 2-A model

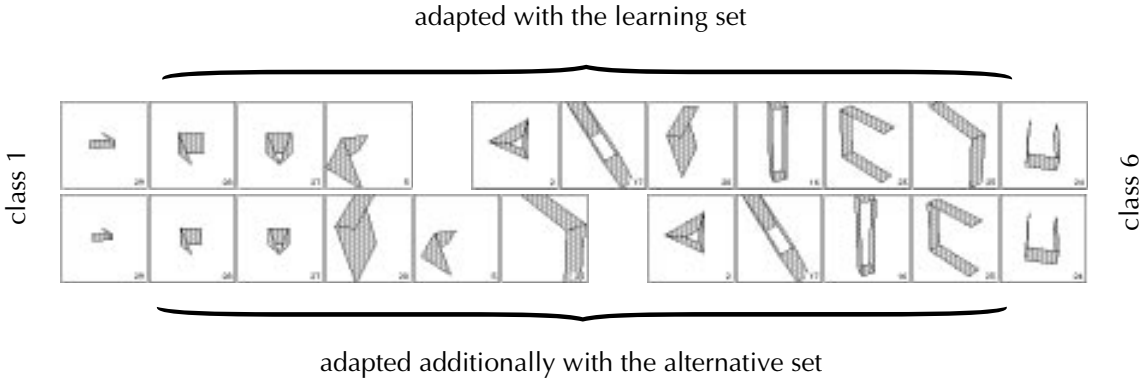

Figure 24: Classification movements

Chapter 12 Realtime Aspects

The *SPIN*-project is designed for the application on a mobile platform. Thus hard constraints regarding robustness and timeliness have to be fulfilled. A possible solution of this demands could be found by employing a formal realtime formalism respectively calculus. Established techniques and references from this domain can be found for example in [Levi90]. But two problems arise when applying such methods. The first one refers to the need of a very strict specification, which is especially hard to formulate regarding realtime constraints. The second one is still the limitation of the project-size, which can be handled with the currently known methods. This prevents definitely the usage of formal techniques (as they are introduced up to now) for the *SPIN*-project, consisting of a large number of parallel and complex processes.

Accepting formal methods not to be adequate for the *SPIN*-project, the author has chosen another strategy for approximating the demands. The whole system (and especially the communication system) should be designed as simple as possible, based on only a few but sufficient and efficient methods. This is only an engineering argument and not a formal technique, but it can help making the system handy and finally "more secure". What this argument means for a single task (e.g. no interrupts or caches etc.) is not illustrated here. Thus a limited communication-system, which gives some very important assurances regarding realtime constraints is discussed in the rest of this section. The proposed communication system is part of the *ALBATROSS* (**A** f**L**exi**B**le multi-t**A**sking and real**T**ime netwo**R**k-**O**perating-**S**y**S**tem kernel), implemented on a VME-bus system and successfully employed by multiple mobile robots. Other aspects of *ALBATROSS*, not discussed in this thesis can be found in [Zimmer91] and [Zimmer92] or following the list of references to other *ALBATROSS* investigations:

- *ALBATROSS* - all the kernel structures: [Wetzler91]
- *ALBATROSS* - the I/O structure: [Peukert91]
- The *ALBATROSS* communication scheduler: [Tebuckhorst92]
- Cyclic scheduling in a realtime communication system: [Peukert94]
- The blackboard communication scheme: [Nölke92]
- A communication monitor: [Sabiwalsky92]

Each of theses works were guided and supervised by the author.

12.1. Concepts of Realtime

Before arguing about realtime constraints, it is necessary to define the meaning of "realtime" in this context. The definitions shown here are based on *time* and on *reliability* aspects and formulated with the background of cyclic scheduling systems.

Concept 1: "Hard Realtime"
A hard realtime task *has to be* fulfilled *every time* in a *fixed* amount of time and other resources. Missing the deadline of a hard realtime task means an "emergency-stop" of the whole system. To fulfil this demand, every kind of time jitter and other uncertainties must be avoided.

Concept 2: "Semi-Hard Realtime"
A semi-hard realtime task *has to be* fulfilled *every time* in an *average* amount of time and other resources. The definition of "average" has to be strict, in the sense of standard deviations or other terms. The deadline of a semi-hard realtime task might be missed several times before emergency measures have to be taken.

Concept 3: "Soft Realtime"
A soft realtime task *should be* fulfilled in time to guarantee the functionality of a system. Missing the deadline of such a task might result in a reduced functionality, but does not refer to security-aspects or the reliability of the system.
In the above definition the *SPIN* components are soft realtime tasks, but the interesting problem is to fit the *SPIN* modules into a hard realtime system like a mobile robot, without disturbing the hard realtime parts, but as an integral part of the whole system. *ALICE*-components may be applied as examples for the other two classes of realtime-tasks.

12.2. Embedding SPIN in a Hard Realtime System

Integrating a system consisting of realtime-tasks at all levels, as defined above on a mobile platform leads to the need of a common communication system, fulfilling several demands. Assuming that the complexity of the system does not allow an integration on one CPU (beside the fact, that such a constellation would arise other problems), an explicit (physical) communication system is required. This is for sure the case for the *SPIN*-system. So the first oncoming aspect is the fact that a system consisting of several asynchronous CPUs has to be synchronized via a communication system, without a significant or, even worse, unpredictable effort for this sychronization. The strict demands are formulated precisely in the following three requirements:

a. **Non-blocking access to the communication-system**
Every call for new data or for an export of new outputs must successfully end within a fixed (and of course short) time - even if one communication partner is broken down or in any other possible constellation.

b. **Passive synchronization**
All the tasks are synchronized *implicitly* by the flow of data, without an explicit trigger (interrupt). Assuming a task in the system is crashed down, the only effect seen by other tasks is the lack of newer data.

c. **Actuality**
The actuality is more important than the order of information, i.e. any kind of flow-control is explicitly ignored on the lowest communication level. The reason for this design decision is that critical information in a hard realtime system is only valid during a small range of time.

A communication system fulfilling these constraints is shown in the next section.

12.2.1. The Realtime Communication Scheme

Before discussing protocols, the hardware-environment has to be defined. In the current *ALBATROSS*-system two independent processors connected via a dual-ported RAM are assumed, i.e. the memory-domains of the processors overlay. The definition of "dual-ported" in this context is given by a configuration where both processors may access the same memory-area *simultaneously*. A conflict at the level of a byte-access has to be solved with special hardware, in a way that neither side is being blocked and a byte will stay an indivisible element.
Following the demands above, a simple communication-buffer access is designed and shown here at the operating system level. The common base of collision-free and locking-free access with two asynchronous partners is a three-buffer structure. In figure 6 the connection between two processors at the hardware-level and the location of the buffers for the communication are shown.

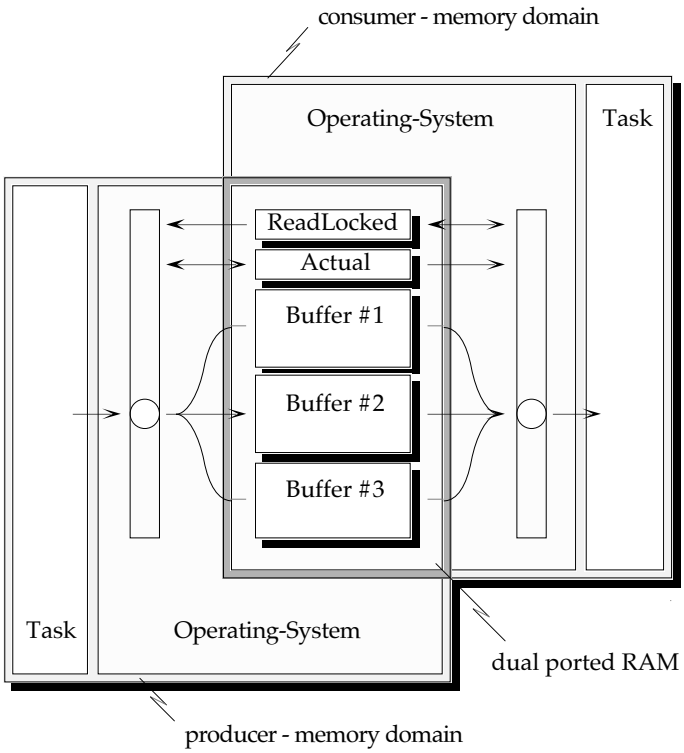

Figure 25: memory domains

In figure 7, figure 8, and figure 9 the implementation fragments (in a Pascal-like syntax) for reading from and writing to the communication-area are shown. The critical accesses to the variables "ReadLocked" and "Actual" are marked (**bold** for a critical writing; *italic* for a critical reading).

Variables not under local control may change their values at *any* time. This fact seems to make any formal proof of the correctness rather hard. Fortunately the critical variables may change only to a small number of values. So all the possible cases can be proven step by step. In order to assure the correctness of the whole realtime-transfer the following four points have to be proven in detail:

- **Mutual exclusion**
  Every communication-buffer is accessed by only one processor at all times.
- **Well-defined results**
  The returned values of each communication-access fulfil the specification always.
- **Termination**
  The execution time of each communication-access is limited by a constant.
- **Actuality**
  The most actual information is being delivered in any case.

```
Type   BufferIndex= (Buffer1, Buffer2, Buffer3);

Var    Actual,       {may only be written by the producer   }
       ReadLocked    {may only be written by the consumer   }
             : BufferIndex;

          {   Both variables have to be one byte long and   }
          {   are located in the dual-ported-RAM area.      }
          {   The initial value of both variables           }
          {   should be "Buffer1".                          }
```

Figure 26: common data-structures

Up to here, only the access-routines at the operating-system level are mentioned. But how does this appear to the task? The chosen syntax is simple and the semantic is much like an electric wire. There are two special functions for each variable, so all possibilities of range- and type-checking may be used. For a consuming task the buffer access is masked by the following interface function:

LookForNew<VarName> (Var <VarName>: <VarType>): Boolean;
e.g.: LookForNewRadarMap (Var RadarMap: RadarShot): Boolean;

The boolean result signals the actuality of the read information (Is this information ever being read before?). For a producing task the buffer-access is hidden by the following interface procedure:

Make<VarName>Available (<VarName>: <VarType>);
e.g.: MakeRadarMapAvailable (RadarMap: RadarShot);

As the final remark in this section the author would like to emphasize once again that reading or writing in this communication-scheme is free of blocking, even when the communication partner has crashed in a critical phase.

```
Type   Actuality = (Old, New);

Function Import (Var ImportedData: Datatype): Actuality;

   Begin
      Import:= Old;

   {--- Phase 1: select a buffer for read-access}
      While ReadLocked ≠ Actual Do
         ReadLocked:= Actual;
         Import:= New
      EndWhile;

   {--- Phase 2: read-access to the selected buffer}
      ReadFrom (ReadLocked, ImportedData)
   EndFunction Import;
```

Figure 27: buffer-access for reading

```
Procedure Export (ExportedData: Datatype);

   VarBuffer, WriteBuffer, CopyOfReadLocked: BufferIndex;

   Begin
   {--- Phase 1: select a buffer for write-access}
      CopyOfReadLocked:= ReadLocked;
      For Buffer:= Buffer1 to Buffer3 Do
         If       (Buffer ≠ Actual) and
                  (Buffer ≠ CopyOfReadLocked) Then
            WriteBuffer:= Buffer
         EndIf
      EndFor;

   {--- Phase 2: write-access to the selected buffer}
      WriteTo (WriteBuffer, ExportedData);

   {--- Phase 3: assign completely written buffer as actual}
      Actual:= WriteBuffer
   EndProcedure Export;
```

Figure 28: buffer-access for writing

12.2.2. Need for a Communication Controller

If the configuration consists of more than two CPUs and the CPUs are not embedded in a local multi-ported RAM-area, which is to the knowledge of the author currently not available at all, there must be an explicit, physical communication system. Thus one of the communication-partners can only access the dual-ported-RAM via a kind of communication system (usually a short range bus-system). This means a break in the realtime-communication scheme shown so far, because the communication is not symmetric at the physical level. One processor may access the dual-ported-RAM much like the local RAM, while the other processor has to use a communication system to access the same dual-ported-RAM.

A new aspect appears from here on. What happens if the access to the (far) dual-ported-RAM fails, because of a disturbance on the communication system? In the above discussion a memory-access was a local transfer and therefore without any aspects of a failed communication.

This problem leads to a contradiction. On the one hand it is necessary to finish a transfer in a short, constant, and predefined time, but on the other hand a failed access via the communication-system must be repeated. The processor for the local task is not available any longer after the first failed trial. So which processor will initiate the second trial? The problem can only be solved by introducing a third processor as a host for the *communication-controller* as shown in figure 10.

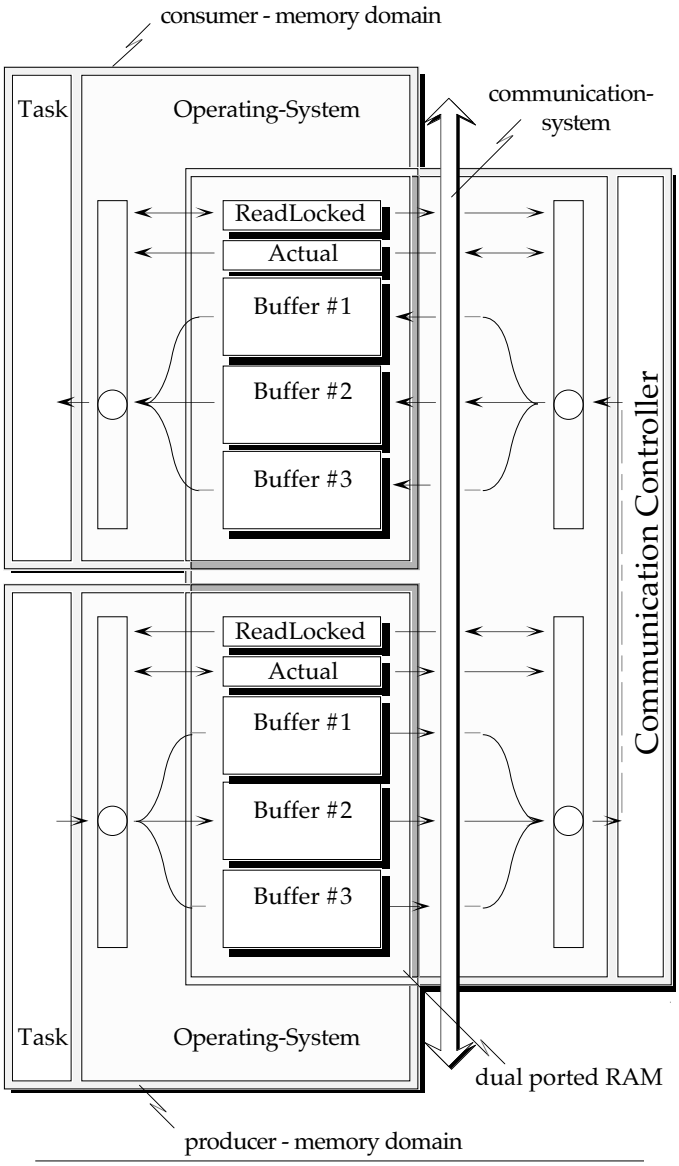

Figure 29: the communication controller

From the view of the tasks any buffer-access looks like an access to the local RAM, i.e. it can be successfully done in a well defined time. The communication-controller may read from or write to the (far) dual-ported-RAMs several times, without disturbing the local tasks at all.

12.2.3. Cyclic Transfers

Cyclic transfers means that there is a pre-scheduling of the communication-slots instead of transient transfers while the system is running. The pre-calculated scheduling plan is then executed in a cyclic manner. The aspects of this kind of transfer will be discussed in the form of three questions:

What is the major problem employing transient transfers?

Whenever a producer wants to distribute its results (this happens completely asynchronously) it runs through an arbitration-phase on the communication-system. The time, the process has to spend in this phase, is quite difficult to calculate – if it can be limited at all.

What is the restriction of pre-scheduling?

The restriction is actually simple – the complete set of communication demands have to be known in advance. In a realtime system the worst case conditions must be calculated and specified. From the worst case conditions the highest needed sampling frequency is being deduced.

What are the advantages of a cyclic-transfer-system?

The construction is based on worst case conditions, i.e. the worst case may happen without any confusion. Moreover, a kind of over-sampling may be introduced. When the maximal needed sampling frequency is calculated, three possibilities have to be considered. First the communication-system is not able to transfer data with such a frequency. So another hardware is needed, or the realtime-constraints have to be relaxed. Second, the communication capacity just corresponds to this highest frequency, so nothing is left to do. But the usual case will be, that there is some extra capacity on the communication-medium, even in the worst, assumed case. In this context "worst case" is defined considering the constraints of the physical system, but all the computer hardware is assumed to be without any failure, especially the communication-system. Thus the extra capacity may be employed to reduce the risk of transient hardware failures by introducing redundant transfers. In a physical sense this may be interpreted as over-sampling.

An example of a hard realtime pre-scheduling system may be found in [Peukert94].

12.2.4. Transfer Synchronization

Up to here a communication-controller and a cyclic-transfer strategy has been introduced, but some problems are still left. These problems will become obvious, when the timing-behaviour of the communication-system as seen by the local tasks is being examined. Two important points should be listed:

- The communication-channels are served with a guaranteed frequency, or even more often.
- The exact time of the update-event (i.e. arrival of new information) on any communication-channel is not explicitly known by the local task (the update-event does not force any asynchronous interrupt).

The typical behaviour of a task in the introduced communication environment will be shown briefly in the following. At some state the task will look for new input-data. If there is new (i.e. not yet processed) information already available, the task will start calculating. Otherwise the task will do some busy-waiting loops on the requested input-channels, until new information has arrived. After finishing the specified work on this new information and having made some output-data available for the rest of the world, the task will check the needed input-channels and the procedure starts again. So, there is an implicit synchronization on the input-data. An explicit specification of the update-time is not necessary.

Unfortunately there are also some other tasks in a realtime-system. Due to correlation purposes, some tasks needs a global time. For example a module composing data from different outer sensors to one representation. Sensor-information from different sources (transmitted via the communication-system), but sampled at the same global time must be fused.

In the following sections two different strategies solving this correlation problem will be introduced.

12.2.4-1 Time-Rigid Transfers

The execution of the scheduling-plan (employing time-rigid transfers) is in the exclusive responsibility of the communication-controller. So the only way to make a global time available is to produce the global time by the communication-controller itself and to embed this global time as an ordinary realtime-transfer in the scheduling-plan.

What are the advantages of time-rigid transfers?

a. The communication-capacity might be used nearly optimal, because the timing of the communication is done by one processor and all the execution-times are exactly predictable.

b. This model implies a high reliability because of:

b-1 The implementation of the communication-controller is trivial, where the execution-phase is addressed here only – *not* the calculation of the scheduling-plan.

b-2 The communication-controller is completely independent from other processors.

c. A global time is available and completely embedded in the whole communication-structure. The global time is exact just at the moment of each update-event on this global-time-channel (see [Levi90] for general time accuracy estimations).

What is the problem of time-rigid transfers?

The global time is quite inaccurate, because the distribution frequency cannot be higher than the maximal frequency from the scheduling-plan. In order to enlarge the accuracy of the global time the local tasks need timers synchronized to this global time. No interrupt is triggered at the update-event on the global time channel. Therefore the tasks are forced to stay in busy-waiting loops on this update-event from time to time (depending on the accuracy of the local timers).

The direct and easiest way to overcome this problem is to implement an asynchronous way of making the global time available – either by an extra hardware or by spending some amount of communication-capacity for this asynchronous time transmission. The second version would lead to a massive violation of the introduced communication paradigm and is not examined here any deeper. The hardware version is proposed by [Kopetz87]. It is assumed that there is no access to a special hardware and the questionable version of asynchronous transfers in the background should not be employed. Thus an alternative synchronization scheme will be shown in the next section.

12.2.4-2 Position-Synchronized Transfers

Assuming that the different correlations between the sampled information in the system are either done by a global time or by a global available position in space (depending on the task), the above synchronization scheme implies two main problems:

- The position-producing task must be synchronized on the global time.
- The position information itself is subjected to an unpredictable time jitter.

In order to overcome these problems two changes in the synchronization-mechanism are necessary:

- The global time and the global position have to be produced by the same task.
- The whole communication system must be synchronized with the combined time-position information.

When the time and the position are produced by the same task, they are correlated automatically, and in the best case this is all done by the communication-controller itself. If the global position cannot be produced by the communication-controller itself for some reason (e.g. no direct connection to the outer world), the communication-controller has to be synchronized on the position-producing task and is therefore no longer independent.

12.2.5. The Blackboard Communication Scheme

Up to this section, the author has shown that the *ALBATROSS*-communication-system is sufficient and efficient with regard of hard realtime constraints. Unfortunately the communication-semantic of a pure electric wire is not satisfying for all the communication demands of the highly distributed *SPIN*-system, even when these communication channels fit perfectly in a hard realtime environment. Thus a whole family of communication protocols is implemented on top of the realtime communication scheme. This is being done without loss of any assurances for the tasks employing the "pure" realtime port concept only. The offered protocols include flow-control, execution-control, broadcasts, client-server-structures, automatic segmentation of infinite data-flows, sliding windows, semaphore, and remote procedure calls. As one example out of this protocol-pool, the blackboard communication scheme will be discussed short in the rest of this section. The blackboard communication scheme is a typical protocol-family employed by multiple entities of the *SPIN*-project. As the name already associates the main concept is a that of a blackboard where multiple clients can read and write simultaneously. The detailed list of feature of this communication schema is shown below:

- Arbitrary number of clients operating simultaneously
- Mutual exclusion (semaphore) in critical operations or operation-sequences
- All operations are client-defined remote procedure calls
- Arbitrary number of types of remote procedure calls per server
- A single remote procedure call is treated as an indivisible unit
- Several status channels per server are offered
- Flow control is applied on all communications

The typical application of the blackboard communication is given by each model out of the "model-bar" of the *SPIN*-project as introduced in chapter 7.2. For a closer discussion of these higher protocols please refer to [Nölke92] respectively [Tebuckhorst92].