Local Learning for Visual Robotic Systems

Florian Hoppe

May 2007

Abstract

In this thesis a new supervised function approximation technique called *Hierarchical Network of Locally Arranged Models* is proposed to aid the development of learning-based visual robotic systems. In a coherent framework the new approach offers various means to create modular solutions to learning problems. It is possible to built up heterogeneous hierarchies so that different subnetworks can rely on different information sources. Modularity is realized by an automatic division of the input space of the target function into local regions where non-redundant models perform the demanded mapping into the output space. The goal is to replace one complex global model by a set of simple local ones. E.g. non-linear functions should be approximated with a number of simple linear models. The advantage of locality is the reduction of complexity: simple local models can more robustly be established and more easily be analyzed. Global validity is ensured by local specialization.

The presented approach relies essentially on two new contributions: means to define the so-called domains of the local models (i.e. the region of their validity) and algorithms to split up the input space in order to achieve good approximation quality. The suggested models for the domains have different flexibility so that the local regions can have various shapes. Two learning algorithms are developed of which the offline version works on a fixed training set that is acquired before the application of the network, while the online version is useful if the network should be continually refined during operation. Both algorithms follow the strategy to place more local models at these regions of the input space where good approximation of the target function is harder to achieve. Furthermore, mechanisms are proposed that unify domains in order to simplify created networks, that define the degree of cooperation and competition between the different local models and that automatically detect data outliers to secure the application of a network. The value of the new approach is validated with public benchmark tests where several competitors are outperformed.

The second major topic of this thesis is the application of the new machine learning technique in an adaptive robot vision system. The task is solved to train an arm robot to play a shape sorter puzzle where blocks have to be inserted into holes. To do so, different software modules are developed that realize interleaving perception-action cycles that drive the robot w.r.t. visual feedback. A visual servoing algorithm is presented that offers a simple principle to learn robot movements. It is based on the acquisition of training samples which represent observations of correct robot moves.

The new approach to machine learning – specifically its features that are uncommon for supervised learning techniques – proves useful to realize this robot vision system. The possibility to combine different information sources in a hierarchy of local models helps to introduce application specific knowledge into the trained models. The outlier detection mechanism triggers error feedback within the system. The online learning algorithm makes the robot system robust against changes of its environment.

Acknowledgments

To be added.

Contents

Chapter 1

Introduction

1.1. Motivation

People like their life to be convenient. So they built machines that do things that have to be done. But things in life are complex and building machines to solve them even more. One can say the workload is shifted from the labor originally performed manually to the development of means for an automation production. And then, engineers like their life to be convenient. So they try to automate the automation. The key idea to tackle this problem is machine learning.

Instead of building highly customized robotic systems that are able to perform specific things, it is ventured on building machines that learn what has to be done. In the different approaches learning spans a continuum between different degrees of autonomy. In the most independent case an artificial system could learn to solve a given task without any human supervision. The system could try certain actions in order to understand what leads to a solution for a stated problem. Less generality in the applied learning strategies is needed if a system can imitate a teacher who shows how humans would do things they want to get done. For such a case the visual performance of the teacher has to be translated to the artificial embodiment. Machine learning becomes even simpler when the desired actions are exemplified by means of the systems' modalities that are used to repeat them. Clearly, target trajectories of an end-effector given in a robot's coordinate system are easier to be processed by machine learning techniques than videos with demonstrations of a human. In the most restricted case many examples would have to be provided by the user to get things done. Then learning degrades to memorizing solutions that have to be shown for all possible situations.

This outline of learning tasks makes clear that the simplification of the development of skillful robotic systems is depending on the degree of autonomy of the learning mechanism. Most preferable are methods that require the least human supervision. The afford to train a system must be well balanced with the resulting competences. The system has to be able to generalize from already experienced situations to new circumstances. Otherwise too many examples of what to do when must be provided by the system designer or the system will not be adaptive enough to successfully act in unrestricted environments. The applied learning techniques must ground a flexibility of the system that can handle robustly more than just changes of the lightening. The dream in robotics is certainly to

built *artificial cognitive systems*.

One first and foremost academical problem with *cognition* is that in the literature no common definition exits for it. Everybody in the research field of machine learning knows what it is about and would agree that it is strongly related to human- or animal-like capabilities to understand the world in order to act in it. But a crisp definition which enumerates necessary and sufficient properties of a cognitive system is not available. Partly because too many people coming from so different areas as neurology, psychology, philosophy, and computer science are discussing various phenomena of cognition. But mainly because cognition as it appears in nature has not been understood yet. If we would know how brains are working in terms of all levels (i.e. physical, biochemical, neurological, psychological), we would certainly better be able to define what engineers try to reproduce. At the moment and for this introduction it must be sufficient to picture the goal with a comparison: the dream of an artificial cognitive system would be a machine that is as adaptive as humans are in order to act in this world.

1.2. Topic of the Thesis

The above outlined goal is just tremendous and this thesis almost necessarily quite limited with respect to it. In the next eight chapters it is suggested how state of the art supervised machine learning techniques can be improved so that the development of artificial cognitive systems is facilitated.

In the relevant area of computer science, learning is understood as function approximation by means of input and output samples. The basic conception is that the capability of a robotic system to act in the world can mathematically be modeled as a continuous multi-dimensional function. The function has to map perceptions to actions. It should relate the state of the world as it is given by sensor data and an internal goal description to commands that steer the robot appropriate to the situation. The repeated application of such a function in so-called perception-action cycles could realize that the system accomplishes tasks over time.

The basic problem with this concept for a cognitive system is that it is practically impossible to model such a function explicitly. The variance in the world is too large that the correct action can be foreseen for every possible situation. The idea for solving this problem is to approximate the function. As mentioned above supervised learning techniques try to achieve this on the basis of pairs of input-output samples of the function. Such techniques construct models (also called networks, estimators or function approximators) for the target function by means of so-called learning algorithms. Depending on the availability of training samples (i.e. input-output pairs) two types of algorithms are distinguished: offline algorithms work on a set of samples , while for an online version training samples become available only separately over time. If the former is applied in a cognitive system, the established model is fixed and the target function must not change since otherwise the computed actions will not fit to encountered situations. In contrast to that, online learning algorithms allow an adaptation of the system to changes of its environment. The price to pay for this desired flexibility is that the construction of a

model in an online scenario is more complicated since every modification can only be based on one training sample.

The demanded capability of a system to be able to generalize from known to new circumstances can easily be transferred to function approximation as a technical approach for learning. A system generalizes better when less training samples are required to approximate the target function with an adequate quality. This connection between the needed training samples and the gained capabilities of the system reveals the crucial point of realizing cognitive systems with machine learning techniques: rather than creating explicit models engineers have to specify input and output spaces of target functions and methods to acquire training samples.

This task is considerably difficult. It would be just naïve if one tries to built a system with one single function that takes raw sensor data such as image pixels as input and is supposed to output control variables for the system's actuators. Such an attempt would certainly fail because the acquisition of the needed training samples would be practically infeasible. A high sampling rate would be necessary since the target function would be highly non-linear. Available supervised machine learning techniques will fail to create a model with too few samples since these would virtually get lost in such a function's input space which could easily reach millions of dimensions.

To overcome such problems one divides the capabilities a system should have into functional blocks that are manageable by machine learning techniques. Different stages of data processing have to be implemented in which the originally available sensor data is transformed into more and more abstract representations. Nature gives a very impressive example of such a cascading information purification. Visual information is processed in the human brain along the so-called visual tract. It starts with neurons in the retina, passes through various layers of the thalamus, and ends in the visual cortex which comprises five different areas. Along this tract the functions realized by small groups of neurons range from comparably simple detection of photons to highly specialized recognition of faces. This gives a notion of what kind of mechanisms are necessary to achieve human-like performance with an artificial system.

Common machine learning techniques do not specifically facilitate such modular solutions. They form solitary blocks which receive inputs and compute outputs without disclosing how latter is done. The internal structure of a trained network typically remains obscure to the system designer. Since the desired model is only implicitly specified with the training set it is hardly possible to introduce available knowledge into its construction. For example the different input dimensions are usually treated in the same manner although often various information sources of the system have to be combined in a functional block. The implicit nature of such models also hinders their interpretation. The dependencies of the different elements inside of a trained network are often incomprehensible. This makes it very hard to realize a proof of concept which guarantees that all possible input values lead to the correct output. Another practical problem with common learning algorithms is that they demand the manual selection of certain parameters. The process of tuning such parameters in order to achieve good approximation quality is mostly quite tedious and relies ofter more on intuition than on clear rules.

To help to overcome these problems new *Hierarchical Network of Locally Arranged*

Models is proposed in this thesis. As a supervised machine learning techniques it offers a coherent framework to create a modular solution to a learning problem. It allows to built up heterogeneous hierarchies in which the different subnetworks can employ different information sources. This helps a system designer to introduce available knowledge into a learned model. The modular nature of a trained network also eases its analysis. This helps to increase the reliability into a built solution.

1.3. Structure of the Thesis

This thesis is divided in two major parts. In the first part, the new approach is described and validated as a generic learning methods that can handle any kind of data, while its application in an actual robot vision system is explained in the second part.

The two parts are organized as follows: in Chapter 2 a survey of the relevant literature about modular machine learning techniques is presented. From this report unsolved problems and interesting ideas are derived that lay the ground for the new approach with its offline and online learning algorithms defined in Chapter 3. The first part is concluded with a chapter explaining various experiments that demonstrate the effectiveness of the new network type.

The topic and common problems of learning-based robot vision systems is introduced in Chapter 5 and exemplified with the so-called COSPAL system in the subsequent chapter. This robot vision systems utilizes – as described Chapter 7 – unique features of the new learning methods that help to introduce domain specific knowledge into a network. The success of the application of the hierarchical network approach in the COSPAL system is proven with experiments summarized in Chapter 8. The very last chapter contains a final summary of the work and concluding remarks.

Parts of the work have been presented in [55, 57, 56, 58].

Part I.

Hierarchical Network of Locally Arranged Models

Chapter 2

Overview of Modular Machine Learning Techniques

Since machine learning techniques (MLT) stems from different research fields such as statistics, neuroinformatics or symbol-based AI, there is no clear and common notion about the question "What is a modular solution to a learning problem?". This is especially true for the more restricted field of non-symbolic supervised learning methods where statisticians might consider this as a question of engineering. Sharkey proposed in [101] a categorization scheme for MLTs that helps to link the different methods described in this chapter. He surveyed so called *Multinet Systems* which use numerous instances of MLTs to approximate a function. He distinguishes *modular* from *ensemble*[1] systems by asking whether "the task in question is decomposed into a number of simpler components" or "several redundant approximations [...] are combined [...] to yield a single unified approach". Following Sharkey's argument, in this chapter methods are presented as modular MLTs that are compositions of a set of non-redundant, comparably simple components.

Sharkey also highlights the difference if the components are combined in a *cooperative* or *competitive* manner. In the first case, a subset of components determines the final output of the MLT. This is typically realized as a weighted sum of the output of individual components. While in the latter case, the control over the output is switched to exactly one component. In addition to Sharkey's definition, one should note that the cooperative or competitive nature of an approach is not just determined by its combination scheme. It is also important how the set of components is established, i.e. to which extend the different components are trained w.r.t. each other.

In the next section Local Linear Model approaches are explained. With their description it becomes clear that the word *locality* can be used as a synonym for modularity. Locality refers to the principle that the different components of a modular MLT are restricted in their responsibility to a local region of the target function's input space. If such components (e.g. linear models) are only locally valid, they are non-redundant. Hence they must be combined to gain a solution that works for the global input space. The local linear model approaches are described in detail since their formalization is comparable with the one of the new MLT proposed in this thesis.

[1] For a description of ensemble techniques such as Stacked Generalization, Bagging or Boosting see [52].

In Section 2.2 the Hierarchical Mixture of Expert approach is described that adds a hierarchical structure to a modular MLT. It is important since it can be conceived as a general framework that subsumes the new approach presented in the next chapter. What follows is a delineation of Self-Organizing Maps that introduces to the concept of locality a neighborhood relation between the single components. This relation is defined on so-called topological maps which base the Local Linear Maps approach. In Section 2.3 this approach is described as a modified mixture of expert network that combines the self-organizing map with local linear models. In Section 2.4 the well-known RBF networks and the Dynamic Cell Structures are outlined. Both use components with "local receptive fields", but only the latter approach restricts the components' responsibility furthermore by a neighborhood graph similar to the topological maps. It follows a short introduction to the classical Decision Tree techniques. These also realize a solution to a learning problem by a division of the input space into local regions. The chapter concludes with final remarks on local machine learning techniques.

2.1. Local Linear Models

A powerful, yet very general idea is to approximate a non-linear function by a number of linear models which are only valid in a local region of the input space. Basically, this is a divide and conquer strategy where the complex global approximation is achieved by combining simple local approximators. Probably due to its simplicity, the idea has a rich history that - according to [28] - goes back to Woolhouse [114] in 1870. Consequently, it was studied in different research fields and hence it is known under several names like "Local Regression", "Piecewise Linear Approximation" or "Locally Weighted Models".

This idea has gained so much interest as it offers advantages due to its two main ingredients: locality and linear models. The former realizes an intuitive problem decomposition that allows a simple interpretation of the gained solution. On the other hand, linear models are profoundly grounded in statistical theory and many powerful and computational efficient methods to determine their parameters are available. In consequence and since linear models are as well easy to interpret, they became an omnipresent tool in probably every quantitative science.

Regardless to their long history, local linear models are still subject to research. The major problem is to define a method that automatically establishes a good task decomposition, i.e. a division of the input space so that linear models can efficiently approximate the target function. Thereby "efficiently" means that as less as possible models approximate the function as good as possible. A subsequent problem of local linear methods raises the question how the local region of validity of a linear model could be defined. An answer has to consider that the flexibility (or rigidity, respectively) of a local region's shape has strong implications on the method to determine a good partitioning of the input space.

In the next subsection, a conceptually simple approach is discussed that exemplifies what kind of solutions one can expect from a local linear model method. Upon this, more complex methods are explained as these are closely related to the work presented in this thesis.

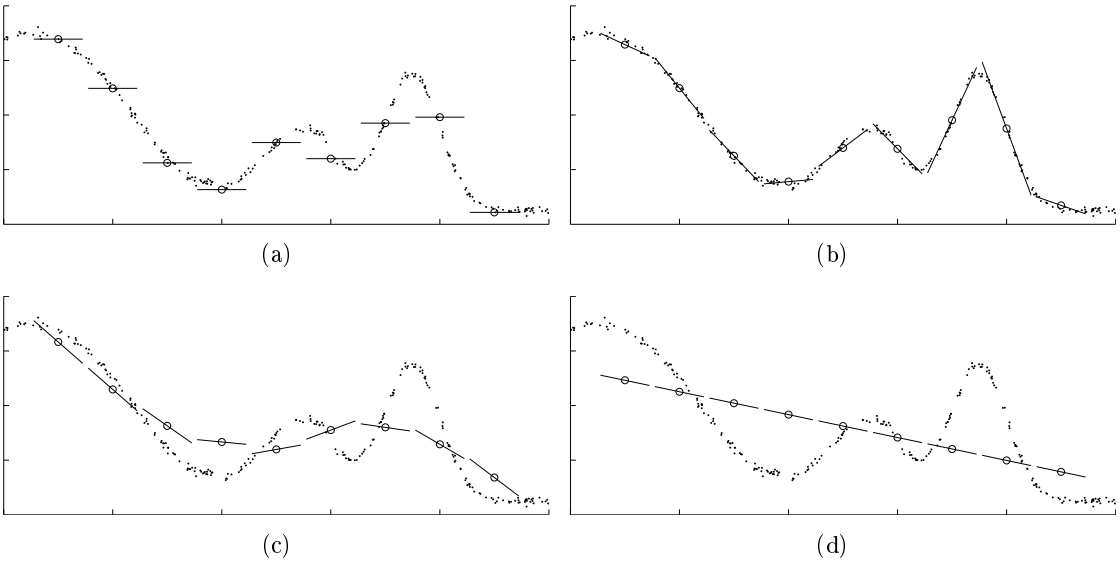


Figure 2.1.: Four approximations of the same synthetic non-linear target function sampled with 250 data points. For panels (a) to (d), different k-nearest neighbor models with $k = \{1, 25, 100, 250\}$ are fitted to the data. In each graph, the linear models are plotted for some input values (marked as circles). Note that the variance between models for different input values decreases from panel (a) to (d). In the extreme case (d), where neighborhood of an input value contains all 250 data points, the resulting linear models do not vary at all.

2.1.1. K-Nearest Neighbor Linear Models

The k-nearest neighbor approach (k-NN) is a very straightforward realization of the local linear model idea (for an overview see e.g. [52]). It solves for each input x the least squares problem:

$$\min_{\alpha(x),\beta(x)} \sum_{i=1}^{k} \Big(y_i - \alpha(x) - \beta(x) x_i \Big)^2, \tag{2.1}$$

where the training samples (x_i, y_i) with $i = 1 \ldots k$ are the k-nearest neighbors to the input x. The set of nearest neighbors is determined by comparing the distances between the input x and all samples from the training set w.r.t. some metric (e.g. the Euclidean distance). The estimated output $\hat{y}$ is consequently given by $\hat{y} = \alpha(x) + \beta(x)x$. Note that with the k-NN approach no explicit model of the target function is determined as the whole training set is stored in order to compute an output value. Hence it is inefficient in both memory and computational costs. This becomes worse with higher dimensional input spaces as a higher sampling rate is needed due to the effect that is known as the curse of dimensionality (see [52, 8]).

With the k-NN approach the possibilities of local linear model solutions can be illustrated very well. By changing the parameter k, one can select the size of the neighborhood around the input x that should influence its estimated output. So with $k = 1$, the k-NN degrades to a look-up table that only gives back the stored output value of the nearest training sample to x. On the other extreme, if k equals the number of samples of the whole training set, the k-NN becomes the global least squares solution which is constant for every possible input value. This explains how bias can be traded for variance (and vice versa) in local linear model approaches (see Figure 2.1 for an illustration). With $k = 1$, one gets estimators with the smallest bias and highest variance. On the other hand, the global least squares solution does not vary, but is heavily biased. Hence, the size of the local region, where a linear model was trained for, has great influence on the possible generalization over the training set. One should also note that the region's size should not be equal in the whole input space. Instead the density of local linear models should be higher in regions where the target function is highly non-linear. Every local linear model approach has to deal with these two facts.

2.1.2. Local Linear Models Approaches based on Domain Models

In contrast to the k-NN approach, methods [99, 106, 68] exist that are explicitly modeling the local region of the input space for that a linear model is valid. This local region is typically named the *domain* of a linear model, a nomenclature that will be used throughout the rest of the text. In the following the approaches of Schaal and Atkeson [99] and Kiong et. al. [68] are described in more details [2] since these are comparable to the new approach presented in this thesis. First the models' definitions are outlined which are quite similar. Afterwards the two learning algorithms are opposed to each other.

The approaches can be formalized as:

$$\hat{f}(x) = \hat{y} = \sum_{k=1}^{M} g_k(x)\beta_k^T \tilde{x}, \tag{2.2}$$

where M is the number of used linear models with the coefficient $\beta_k \in \mathbb{R}^{n+1}$ and $\tilde{x} = (x, 1)^T \in \mathbb{R}^{n+1}$ is the extended input x that allows a constant term in the linear equations. The domain of the k^{th} linear model is defined with the weighting function $g_k(\cdot) \in \mathbb{R}$. Only this definition is specific to the different approaches. The common idea is that depending on the distance of the input x to the domain, the weighting factors are larger or smaller and hence, will weight the output $\beta_k^T \tilde{x}$ differently. It is part of the concept that more than one linear model contribute to the output $\hat{y}$ (i.e. more than one $g_k(x)$ is unequal zero). This means that the domains are overlapping each other to a certain extend. Furthermore the weighting functions are normalized, i.e.: $\sum_k g_k(x) = 1$ and $\forall k : 0 \leq g_k(x) \leq 1$. This realizes a partition of unity of the input space, without any holes where all weighting functions are equal zero.

[2] The approach of Tagscherer et. al. [106] is not well enough documented for that purpose.

Basically, both methods define a model's domain by calculating the weighting factors according to the Gaussian function with the mean vector $\mu_k \in \mathbb{R}^n$ and the matrix $D \in \mathbb{R}^n \times \mathbb{R}^n$:

$$g_k(x) = \exp\Big(-\frac{1}{2}(x-\mu_k)^T D_k (x-\mu_k)\Big). \tag{2.3}$$

Regarding the equipotential lines of $g_k(\cdot)$, such a domain can be interpreted as a hyper-ellipsoid centered at μ_k with size and orientation given by D_k. Kiong et. al. restrict the matrix D to be diagonal. Hence the axes of the resulting hyper-ellipsoid are parallel to the axis of the input space. Furthermore these authors extended (2.3). As a so-called *expertise level*, they multiply the sigmoid function $\alpha_k = 1/(1+e^{-\gamma_k})$ with $g_k(x)$. With the scalar parameter γ_k, the authors want to model the trustworthiness of the k^{th} linear model independently from the input x.

Different learning algorithms were proposed to determine appropriate values for the models' parameters. In a straightforward manner, both approaches compute the parameters β_k of the linear models by minimizing the weighted least squares error functions [75]:

$$J_k = \frac{1}{N}\sum_{i=1}^{N} g_k(x_i)\Big(y_i - \beta_k \tilde{x}_i\Big)^2, \tag{2.4}$$

where N is the number of all training samples (x_i, y_i). But different strategies are used to compute values for the free parameters M, μ_k, D_k of the weighting functions $g_k(\cdot)$ and γ_k.

The parameter M, defining the structure of the estimator $\hat{f}(\cdot)$, is incrementally determined by inserting and pruning linear models. Given a training sample (x_i, y_i), Schaal and Atkeson suggest to add a new model k if no weighting factor $g_k(x_i)$ is larger than a manually chosen threshold. The new domain is positioned at the training sample (i.e. $\mu = x_i$) and the matrix D is set to a certain initial value. On the other hand, a linear model is pruned if two domains have too much overlap. This is the case if the weighting functions of two domains are larger than another threshold. Then the linear model which matrix D has the larger determinant is removed.

Similar to that, Kiong et. al. are also working with different thresholds to find an appropriate number of models. They just ground insertion and pruning decisions on different parameters of their model: in addition to the weighting factor $g_k(x_i)$, the insertion is also depending on the performance of the estimator (mean squared error over training samples). Whereas pruning is performed when the expertise level α_k of a linear model falls below a certain value.

More clearly, the approaches differ in their way to adjust the position μ_k and shape D_k of the domain. A two-phase learning scheme is proposed by Kiong et. al.: the center vectors are changed with Hebbian adaptation steps [8, 53] and by a gradient decent approach to minimize the least squares error function over the training set. Also the expertise level parameter γ_k is trained in this step. In contrast to that, Schaal and Atkeson prefer to minimize a so-called locally weighted error function which emphasizes that training samples only effect the domain's parameter that they belong to. As discussed

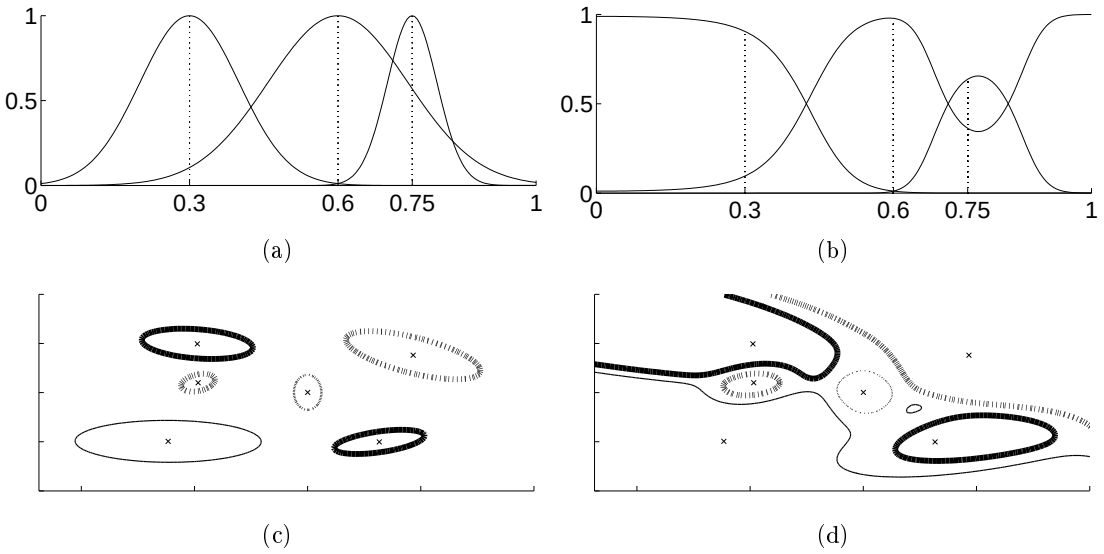


Figure 2.2.: Effects of normalization: the upper row panels shows three Gaussian functions with different means and standard deviation. The lower row shows the equipotential lines of two-dimensional RBFs. In the panels (a) and (c) the functions are plotted without, and in the panels (b) and (d) with normalization. The distortion of the shape of the functions is clearly visible. Note that at the right border of graph (b), the middle Gaussian suppresses the right one. The same effect is visible in panel (d) with a small peak near the middle that belongs to the lower left function.

in [64, 79] the latter suits better the overall strategy of local models since promotion of competition between different models improves their local approximation performance. Due to the use of gradient decent approaches, for both learning algorithms one has to specify various parameters for learning rates, regularization and momentum terms. The methods of Schaal and Atkeson [99] and Kiong et. al. have 6 and 9 free parameters, respectively. These have to be manually fine tuned for a specific application.

In summary, one should note several points about these two local linear models. They built a much more compact form to approximate a target function than the k-NN approach. With their fixed and typically small number of linear models, they offer great computational advantages since not the whole training set has to be stored and processed to compute one output value.

The compactness promotes also that the solutions can be interpreted. It follows from the domain definition given by (2.3) that the k^{th} linear model has the most influence on the estimated output $\hat{y}(x)$ at the maximum of $g_k(\cdot)$. That is reached when x is equal to μ_k. The influence decreases radial symmetrically according to the matrix D. This should give a clear picture of the region of the input space where the individual linear models most prominently determine the output.

Unfortunately, the normalization of all weighting functions $g_k(\cdot)$ makes things more

complicated. In [103] quite striking side effects of normalization on radial basis functions are discussed. In short, the domains are significantly distorted. Figure 2.2 illustrates different effects. Normalization causes that equipotential lines of a radial basis function are no longer hyper-elliptical. Their maximum can be shifted from the original mean vectors. And maybe worst, the values of $g_k(x)$ may not decrease monotonically with increasing distance between x and μ_k. This means that the weighting functions may become multi-modal (see panel (c) and (d) in Figure 2.2), and hence a domain may be torn apart. That contradicts the idea of local linear approximation. These effects are caused by the fact that the shape of single domain is more influenced by its relation to other domains than by its own parameters. That complicates the interpretation of the means μ_k and matrices D_k considerably.

Finally, note that due to the domain definition (2.3) the function approximator $\hat{f}(\cdot)$ is a continuous function of x. No jump discontinuity in the output will occur if one follows a continuous trajectory through the input space. That is not the case for the k-NN approach. Moving through the input space will result in the selection of different training samples as the k-nearest neighbors. Hence quite different linear models can be computed for quite similar inputs. This results in rather jumpy output values. The effect will be most prominent when the sampling rate is low, but the non-linearity of the target function is high. Nevertheless, whether this is an major drawback of the k-NN approach depends on the application. If the input is not changed continuously, there is no problem with discontinuities in the output as quite different output values can be expected for quite different input values. On the other hand, the discussed parametric local linear model approaches have problems if the target function itself has discontinuities. This is especially true for classification problems since the output values are discrete by definition.

2.2. Mixture of Experts

In the research field of Artificial Neural Networks the *Hierarchical Mixture of Experts* (HME) approach has a long and profound history. Proposed by Jordan et al. [62] in 1991 (see [64] for a nice summary), the HME was and still is successfully applied to various real-world problems, e.g. multi speaker vowel recognition [61], object recognition [11], speech recognition [43], the design of compensators for intensity modulated radiation therapy [46] and speaker identification [25]. Many variations and improvements (e.g. [111, 5, 94, 102, 107]) of the original idea were proposed. The HME approach offers that one can structure the solution to a learning problem in a hierarchical and modularized way. The idea is simple but compelling: a set of so called *expert networks* is combined by a *gating network* to a possible multi-layer hierarchy of specialized networks. Figure 2.3 exemplifies the architecture. As a function of the input, the gating network should combine the output of the expert networks to compute the overall output of the hierarchy. The experts itself are meant to be specialized, i.e. trained to predict the correct output not for all but for special input values. It is part of the original idea that both the gating and the expert networks can be any appropriate machine learning technique. Besides the simplicity of

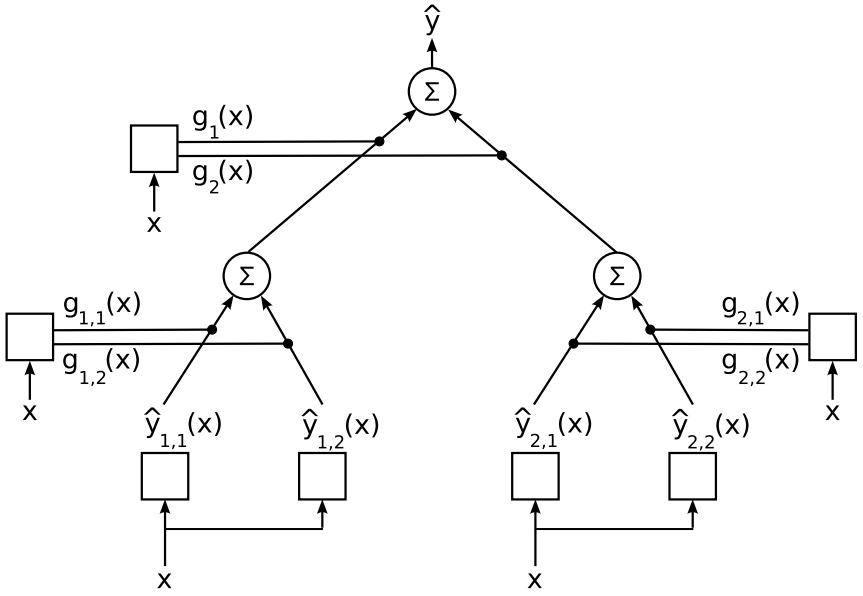


Figure 2.3.: A two-level *Mixture of Expert Hierarchy* with three gating networks and four experts networks.

the idea, this is the reason why the HME framework has so many descendants.

The generality of the HME is based on its formalization. The two-level hierarchy depicted in Figure 2.3 would be defined as:

$$\hat{f}(x) = \sum_{i=1}^{2} g_i(x) \sum_{j=1}^{2} g_{i,j}(x)\hat{y}_{i,j}(x), \tag{2.5}$$

where the expert networks of the different levels are given by $\hat{y}_{i,j}(\cdot)$, and the gating networks of the first and second layer are the scalar functions $g_i(\cdot)$ and $g_{i,j}(\cdot)$, respectively. Obviously, the parametric local linear models presented in the last section can be viewed as an one-level mixture of expert network: the M weighting functions $g_k(\cdot)$ are forming the gating network and the expert networks are defined by $\hat{y}_k(x) = \beta_k^T \tilde{x}$. The authors Jacob and Jordan suggested different definitions for the gating and the expert networks (compare [62, 61, 64]). The most recent and probably best known one is presented in [64] where, for regression problems, the expert networks are given as linear models.[3] Furthermore, the gating factors $g_i(\cdot)$ are based on a linear combination of the input x:

$$g_i(x) = \frac{\exp\left(v_i^T \tilde{x}\right)}{\sum_k \exp\left(v_k^T \tilde{x}\right)}, \tag{2.6}$$

where v_k is a weight vector for each $g_k(\cdot)$ of one gating network. The definition realizes a *softmax* normalization [13] which divides the input space along linear "soft" margins.

[3] This is equivalent to the local linear model approaches.

The resulting partitioning is called "soft" since with the summation in (2.5) and the normalization of (2.6) with a logistic function, the output values of the expert networks are blended together (hence, the name "mixture" of expert). The "softness" and direction of a margin is determined by the magnitude and direction of the weight vectors v_k. With this definition, each gating network induces a smooth planar partitioning of the input space where lower-level gating networks are dividing the partitioning of the higher-level networks into smaller regions.

The original authors are interpreting their hierarchy as a probabilistic decision process that determines a linear mapping from the input to the output space. The weighting functions $g_k(\cdot)$ are viewed as random variables deciding which expert has to produce the final output. This interpretation grounds a learning algorithm for hierarchies with fixed structure (i.e. fixed number of experts and layers of gating networks). The algorithm is an EM-Algorithm [32] that showed fast convergence rates on artificial data sets. The major drawback of the implementation is that the model selection problem is left to the user. One has to decide about the structure of the hierarchy. The appropriate number of expert and gating networks is not automatically determined.

Note that the gating and the expert networks are serving different purposes using the same information source. The gating networks compute appropriate weighting factors, while the experts are performing the actual mapping between input and output space. Still they are defined as functions of the same input x. To our knowledge, this discrepancy has never been discussed. A straightforward relaxation is to allow different inputs for the two type of networks. We argue in Subsection 3.1.4 that the exploitation of different information sources by the different networks in a hierarchy offers significant potential to improve performance.

2.3. Self-Organizing and Local Linear Maps

Another well-known method that relies on a locality concept is the *Self-Organizing Map* (SOM) approach of Kohonen [71, 72]. It is an unsupervised learning technique. It realizes a mapping from a high-dimensional input space into a low-dimensional coordinate system, the so called *topological map*. This map, typically an one- or two-dimensional grid, should represent a data manifold in the input space. In the SOM model, the manifold is given as a set of vectors, each is an element of the input space and associated to a node of the topological map. This way, such a vector represents as a prototype its local neighborhood in the input space. With a nearest neighbor decision, this induces a Voronoi tessellation of the input space.

During the training of a SOM two different types of neighborhood measurements are coupled. One real valued distance in the input space, one integer valued distance on the topological map. The former is typically the Euclidean distance. The latter states the minimal path length (i.e. number of crossed edges) from one node to another on the map. The position p_k of some prototype k is shifted toward a new training sample x_i according to:

$$p_k \quad \leftarrow \quad p_k + \alpha(x_i - p_k), \tag{2.7}$$

where $\alpha \in (0, 1]$ is a manually chosen learning rate. This learning algorithm works similar to an online version of the k-means clustering method [76, 8]. They differ w.r.t. the choice which prototypes are updated. Applying a "winner takes it all" principle, the common k-means method moves only the prototype closest to the input sample. While the SOM algorithm also shifts those prototypes that are w.r.t. the topological map distance neighbors to the "winning" one. Less technical speaking, pulling at one node has the desired effect is that its neighbors on the grid are also fastened over the data manifold.

Local Linear Maps (LL-Map) were proposed as an extension of SOMs in [97, 96]. LL-Maps are function approximators trained in a supervised learning scheme. The basic idea is to associate to each node of a normal SOM a locally valid linear mapping from the input to the output space. An output reference vector $w_k \in \mathbb{R}^m$ and a $n \times m$ matrix A_k is assigned to each node k, so that the output of a LL-Map is given by:

$$\hat{f}(x) = w_s + A_s(x - p_s), \tag{2.8}$$

where the "winning" node s is determined with $s = \arg_k \min \|x - p_k\|$. So, a correction to the node's output reference vector w_k is computed that is linear in the deviation of input x from its prototype p_k. Like in the k-NN approach, the realized output function $\hat{f}(\cdot)$ has discontinuities at the borders of the induced Voronoi tessellation cells. This results from the applied "winner takes it all" principle. To overcome the problem, a LL-Map can be modified to be:

$$\hat{f}(x) = \sum_k g_k(x)\Big(w_k + A_k(x - p_k)\Big), \tag{2.9}$$

where $g_k(\cdot)$ is a "soft-max"-function

$$g_k(x) = \frac{\exp(-\|x - p_k\|)}{\sum_i \exp\left(-\|x - p_i\|\right)}. \tag{2.10}$$

Then the similarity to an one-layer Mixture of Expert network and the local linear model approaches becomes obvious. In the former case, the gating network is realized by a SOM. Compared to the latter ones, LL-Maps offer less flexibility to model the local domains since the matrix D from (2.3) is omitted in (2.10). The main differences remain in the learning algorithm: the positions of the prototypes p_k are determined with the SOM method, while the output reference vector w_k and the matrix A_k are updated according to:

$$w_k \leftarrow w_k + \beta\Big(y_i - \hat{f}(x_i)\Big) + A \cdot \|x_i - p_k\| \tag{2.11}$$

$$A_k \leftarrow A_k + \gamma(y_i - \hat{f}(x_i))\frac{x_i^T}{\|x_i\|^2}, \tag{2.12}$$

where (x_i, y_i) is one training sample and β and γ are learning rates.

The key idea of the SOM and hence LL-Map method is to build up neighborhoods by connecting data prototypes via edges. Those edges are establishing a distance measurement based in a different space than the input space. Hence a different notion of locality

is gained for data of the original input space. This idea is exploited in *Dynamic Cell Structures* (discussed in the next section) and in this thesis as described in Section 3.6.

2.4. RBF Networks and Dynamic Cell Structures

Radial Basis Function (RBF) networks [17, 78] realize a locality concept that is different to the methods presented in the last sections. They do not define local modules like e.g. the HME approach that restrict the responsibility of whole subnetworks to a local region of the input space. Instead RBF networks rely on certain units that are often termed "neurons with a local receptive fields". They approximate the target function as a linear combination of basis functions. Each basis function can realize localization with a *kernel* function $K_\sigma(x, \mu)$ that assigns a weight to x based on the distance to μ and a scale parameter σ. With the most often used Gaussian kernel function, a RBF network with M basis functions is given by:

$$\hat{f}(x) = \sum_{k=1}^{M} \beta_k \exp\Big(- \frac{\|x - \mu_k\|^2}{2\sigma_k^2} \Big) + \beta_0, \tag{2.13}$$

where $\beta_{0,1,\ldots,M}$ are the coefficients to linearly combine the basis functions. A RBF network can be described as a feed-forward neural network that contains two different layers. The first one, defined with the kernel function, has M hidden units whose outputs are linearly weighted by the second layer to compute the final output.

Such a network is typically trained by minimizing the mean squared error (MSE) over a training set. Without constraints on the parameters $\{M, \mu_{1,\ldots,M}, \sigma_{1,\ldots,M}, \beta_{0,\ldots,M}\}$, the MSE criterion is non-convex and has multiple local minimal, hence needs a non-linear optimization technique. To ease the problem, the training of a RBF network is often divided in two phases. First, the parameters of the kernel functions are determined, then the coefficients β_k can simply be estimated by a least squares method. But still the right choice for the kernel functions (their number, center and scale parameters) remains as crucial as for every other method that relies on a partitioning of the input space into local regions.

Dynamic Cell Structures (DCS) [20, 19] combine RBF networks with the idea of using edges to represent neighborhoods. The center vectors μ of a RBF network are understood as vertices in a neighborhood graph that connects adjacent local regions. In contrast to the SOM approach, the graph is not a fixed grid but solely dependent on the data distribution between the center vectors: given a training sample (x, y), an edge is drawn between the two vertices with the smallest and second smallest Euclidean distance to the input x. The generated graph restricts the number of kernel functions that contribute to the output of the whole network, so that

$$\hat{f}(x) = \sum_{k \in A(x)} \beta_k \exp\Big(- \frac{\|x - \mu_k\|^2}{2\sigma_k^2} \Big), \tag{2.14}$$

where $A(x) \subseteq \{1, \ldots, M\}$ is the set of vertices that contains the "winning" vertex

$s = \arg_k \min \|x - \mu_k\|$ and its direct neighbors in the graph. This way, the DCS methods intensifies the locality constraint in a solution to a function approximation problem.

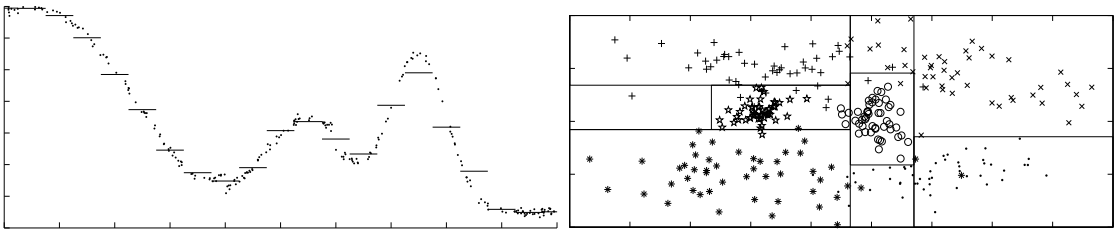
Grounded on the neighborhood graph, strategies to train a DCS were proposed in [19]. Like for RBF networks, the learning algorithm is split up into two phases where in the second one, the coefficients β_k are still determined by a least squares method. While in the first phase the number and position parameters of the kernel functions are adapted by utilizing the neighborhood graph. An appropriate number of kernel functions is estimated with an iterative scheme that adds new ones if the approximation performance does not meet a certain threshold. According to one option proposed in [19], a new center vector is placed between the two centers which are closest to the training sample and adjacent to each other in the graph. This option should be more robust to noisy data than the method where a new center is positioned directly onto the training sample (compared with the insertion strategies in Subsection 2.1.2). The adaptation of already existing center vectors is comparable to the method for SOMs and LL-Maps: only the center vector of a "winning" vertex and its neighbors in the graph are shifted towards a training sample. This should ensure that those units of the network that will dominate the output afterwards are optimized for their local region of the input space. The main difference between the two approaches remains in the origin of the graph or topological map, respectively. The DCS graph is generated purely data driven. In contrast to that for SOMs, one has to specified the number of nodes and the dimensionality of the applied map. If no knowledge about the intrinsic dimensionality of the processed data is available, the DCS approach has clear advantages.

2.5. Classification and Regression Trees

To solve a learning problem, *Classification and Regression Tree* (CART) techniques also follow the strategy to divide up an input space into local regions. But since the used representation and the needed learning algorithms are quite different to these of the approaches described above, only the basic principles of CARTs are outlined in this section. For a comprehensive summary of CART techniques the reader is referred to [95] or to Quinlan [93] who proposed the probably best known implementation of a CART, the C4.5 algorithm.

In contrast to all the methods described above, the regions of CARTs have sharp boundaries, i.e. either a sample belongs or does not belong to one region. This stems from the fact, that CART methods are mainly used for classification problems, where one class label is assigned to each local region to be the output for the data that falls inside of it. When applied to a regression problem, the average output value of the training samples inside a region is typically given as output.

CART methods are also termed *Decision Trees* since the question to which region a data sample belongs to is decided with a tree-structured model representation: samples are passed down a typically binary tree, with decisions being made at each node until a leaf of the tree is reached. At each inner node, the value of one input dimension of the sample is examined (e.g. $x_1 \leq 5$) to decide in which subtree the sample has to be processed further. When the decision process terminates at a leaf node the associated



(a) (b)

Figure 2.4.: Effects of CART: panel (a) shows that a regression tree approximates a target function in a stepwise manner. In panel (b), a synthetic 2D classification problem with six different classes is depicted. The rectangles show how a CART would divide the input space, each of them correspond to a leaf node.

class label (or the continuous output value, respectively) is given as output. Note that the optimal classification tree would be perfectly balanced i.e. each class would be represented by exactly one leaf node. With trees containing more nodes, more decisions would be necessary to classify a sample.

CART learning algorithms construct such trees by successively splitting the training set. Each split adds an inner node to the tree. Later pruning of nodes helps to keep a tree well balanced. Several splitting and pruning rules have been proposed (see [12] for examples). The general idea of the various splitting rules is to reduce a special "impurity" measure that quantifies how many samples of different classes are represented by one node, i.e. how "pure" a node is. On the other hand, pruning techniques try to keep a good proportion between performance and size of the tree.

When comparing CART with the other local methods, one should note two different points. The local regions induced by a basic CART are hyper-rectangles parallel to the axis of the input space with sharp boundaries. The other methods realize regions with soft boundaries that are shaped like hyper-ellipsoids or due to normalization are flexible non-linear functions. On the other hand, regression trees are approximating continuous target functions typically as stepwise constant functions. Both effects are illustrated in Figure 2.4.

2.6. Summary

In this chapter non-symbolic learning techniques were presented that follow a modular approach. It was taken the view that modular means that the solution to a learning problem is a composition of a set of non-redundant, comparably simple components. The components are non-redundant as they govern the output only in local regions of the input space. Hence the reviewed learning techniques realize modularity by the locality of their different components.

Local linear methods are approximating a target function by a set of simple linear

models that are only valid in local regions of the input space. The k-NN approach is a memory-based technique where locality is defined by the k nearest training samples to a given input. More advanced methods built more compact solutions which use explicit domain models to restrict the validity of a single linear model. Typically, such domain models are based on normalized Gaussian functions.

The Hierarchical Mixture of Expert approach can be viewed as a general framework that offers a convenient notion of modularity. It distinguishes expert networks from gating networks that work on different hierarchical levels. Practically, it has the disadvantages that the structure of the hierarchy has to be defined manually and that the input space is divided by linear margins which have not local but global effects.

Originally proposed for dimension reduction, Self-Organizing Maps were extended to Local Linear Maps to solve regression problems. The key idea is to constrain the process of partitioning of the input space by coupling a metric for the input space with a metric for the employed map. Again, one has to chose the structure of this map a priori.

In Radial Basis Function Networks and Dynamic Cell Structures locality is realized by kernel functions that define local receptive fields. They perform a mapping to the output space not directly from the input space but from the space induced by the kernel functions. In contrast to RBF networks, the DCS approach restricts the locality by a neighborhood graph that is established by a purely data driven process.

Finally, Classification and Regression Trees were described that classically divide the input space in hyper-rectangles. The learning algorithms recursively split the training set w.r.t. to a specific impurity measure. These techniques have typically the disadvantage that they produce only stepwise constant approximations for regression problems.

Chapter 3

Hierarchical Network of Locally Arranged Models

In this chapter the new Hierarchical Network of Locally Arranged Models (HLAM) is presented. It is a supervised machine learning technique. The goal of its development was to realize a modular approach that improves ideas of the state of the art techniques that were presented in the last chapter. The new method is introduced with an outline of its principles and main features in Section 3.1. The first section addresses the basic challenges of the approach and prepares the reader for the discussions in the rest of this chapter. In the different sections alternative solutions to certain subproblems will be proposed. So altogether, a kind of construction kit is offered where one can decide by which means an HLAM can be built for a specific application at hand. How successful the new approach with its different alternative solutions can work is validated with a number of experiments in the next chapter.

3.1. Principles and Features of HLAM

This overview section is structured as follows: first the idea and the basic principles of the HLAM approach will be introduced and motivated. Upon this, the challenges to construct an HLAM will be explained. To do so, two basic questions that ground the actual problems are discussed and answered. This is done in an informal way but the added cross references to the proper definitions should guide the reader through the whole chapter. Only the first basic definitions will be given in Subsection 3.1.3. This introductive section will be concluded with the description of the more advanced features of the HLAM approach.

3.1.1. Basic Principles

As a supervised learning technique, the basic goal of a Hierarchical Network of Locally Arranged Models is to approximate a target function. The target function is implicitly given as a set of corresponding samples from an input and an output space. Similar to approaches mentioned in Chapter 2, the network consists of a set of individual non-redundant components. These components are individual models that are specialized to

approximate the target function in a local region of the input space. Such a local region is in the following referred to as the *domain* of a so-called *local model*. The models itself can be any kind of mapping between the input and the output space that can be trained in a supervised manner.

So, the first key principle of the HLAM approach is that the individual models are locally arranged in the input space, i.e. they are only valid in their domain. That realizes a division of the input space that structures the solution to a given learning problem. The second principle is that this decomposition can be done in a hierarchical manner. This establishes subdivisions of domains, so that similar but still different models are grouped together. These local models are different as they are specialized to their domain. But they are also similar as their domains can be covered by a domain from a higher level of the hierarchy.

These two principles offer to reduce the complexity in first creating and afterwards understanding a solution to a learning problem. Since the local models have to be accurate only in their restricted domain, rather simple machine learning methods can be employed for them. So, for example a highly non-linear target function can be approximated by a set of polynomials of low degree or even simple linear models (see Figure 3.1). On the other hand, the division of the input space into local regions helps to understand a realized solution. For each domain one can separately analyze the local models in order to see how they map data from the input into the output space. E.g. the global stability of an adaptive controller can be proven by applying knowledge from control theory to the local models (as done in [80]).

The reduction of complexity when creating local models also simplifies the interpretability of the whole network. Since the local models are restricted to their domain they can be trained with simpler methods (e.g. least squares regression for linear models instead of error back-propagation for multi-layer perceptrons), and hence the result is easier to analyze. Furthermore, the hierarchical structure of the network helps to break down a learning problem into smaller and smaller subproblems. Each level of a hierarchy defines how domains are divided into subdomains. They represent different abstraction levels. Data belonging to a certain subdomain belongs automatically to the domains from higher levels of the hierarchy. Hence the models that handle the data from the subdomains are grouped together by a next higher level domain. This principle is a process of abstraction that makes an HLAM amenable to fruitful interpretation.

3.1.2. Basic Challenges

To realize such a modular approach outlined above, one has to answer two major questions:

1. How is the domain of a local model defined?

2. How is the input space split up into a set of domains?

An answer to the first question conditions the possible shapes of the local region that is governed by a local model. Consecutively, the shapes of the domains strongly influence

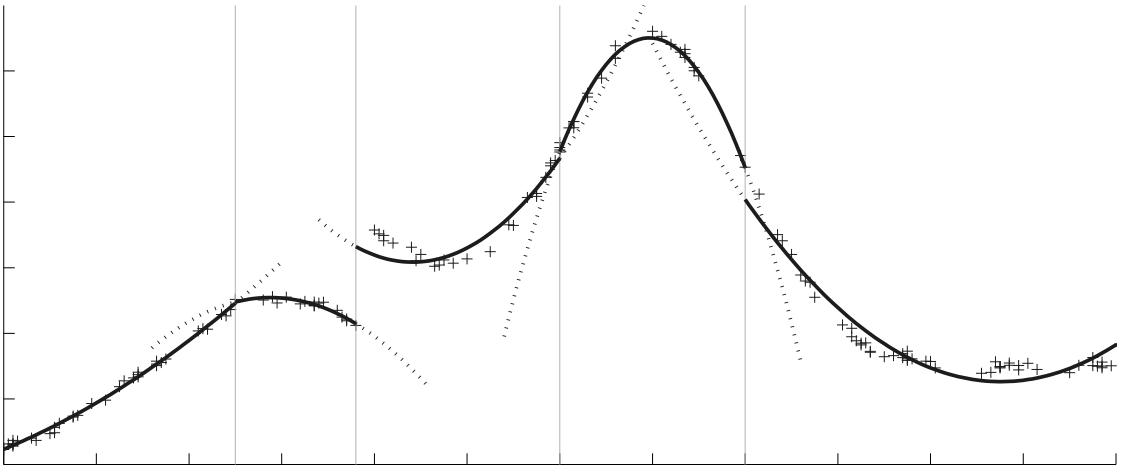


Figure 3.1.: Approximation of a non-linear, discontinuous target function with a set of only locally valid polynomials of 2^{nd} order. The training sample are shown as $+$. The polynomials are plotted as connected lines in their own domain and dotted outside of it. The vertical lines indicate the boundary between two adjacent domains.

the number and approximation performance of the local models. So, the final goal is to find a good compromise between a flexible but still simple domain model. Given such a model, one can decide on a strategy to divide the input space in order to answer the second question. The needed process should meet two main goals: the achieved approximation performance of the local models should be as good and their number as small as possible. Since these two goals are strongly dependent on each other, again a good compromise has to be established to fulfill both. The next two sections give a detailed discussion on these problems along with first outlines of the their solutions.

3.1.2.1. On modeling a domain

If a domain's shape is highly restricted - maybe to be a hyper-cube in the input space of fixed side length and orientation - it is most likely that many local models are needed to approximate the target function. It is also quite likely that local models assigned to adjacent domains are identical. This will happen when a local model can approximate the target function in a larger region of the input space than the restrictive domain model can enclose. Both effects – a high number of models and duplicates – are not desirable. They increase the computational workload in time for deciding which model is used and in space for storing identical model parameters. Furthermore the interpretability of a network suffers with too many local models. In such cases, two strategies for improvement are reasonable: either a set of adjacent domains are combined so that one single local model can be assigned to this enlarged domain. Or one relaxes the conditions on the possible shapes of a domain.

On the other hand, the danger of a too flexible domain model is twofold. The inter-

pretability of an HLAM will degrade if the single domains can hardly be understood. If a domain model is too complicated, the question whether a data point belongs to or does not belong to a certain domain is difficult to answer. In such a case one can hardly draw conclusions about a built network. Secondly, a complex domain model that can adopt very different shapes is typically computational very demanding. Parameter estimation can become a problem especially if not enough training data is available. Another challenge might be the necessity to fine tune the meta-parameters of this estimation process.

Besides the last mentioned drawbacks, more flexible domain models are commonly still preferable to simpler ones. After all, the basic idea is to divide the input space into large regions that can be handled by simple local models. If there exists such a local region where the target function can be approximated by just one simple model, then this region should be enclosed by one domain model. That is due to the fact that in a larger region – even if it has a rather complicated shape – typically more data is available to train the local model. That increases the robustness of the model and hence the performance of the whole network. So, by deciding on the complexity of the domain models, one can trade performance with ease of interpretability. And commonly, the performance of a machine learning technique is regarded as more important.

The discussed issues are illustrated in Figure 3.2 and Figure 3.3. The first one shows in two graphs identical training data in a 2D input space, and how these are enclosed by two different domain models. Three different symbols (o, + and ·) indicate that the different samples should belong to three different local model. This means that the best solution of an HLAM to this synthetic example would be a perfect separation of these three sample sets. In the ideal case, three domains would enclose the different sets, and only three different models would be needed to perform an accurate approximation of the target function (which can not be shown in the graphs).

In panel (a) of Figure 3.2 a solution with the center domain (CD) model that is explained in Subsection 3.3.1 is shown. Such a domain model consists of a prototype vector which is the mean value of all the samples belonging to a domain. With a nearest neighbor decision the CD model results in the Voronoi tessellation of the input space. In the example four domains with their prototypes (their positions are indicated with *) were defined, and the boundaries between them are shown in the figure.

One should note two things: a perfect separation of this data is not possible with just three CD domains. The shown solution contains two identical models (for the data marked with an o). This domain model is too inflexible to capture this data with an appropriate number of domains. Secondly, such domains can be unlimited. In the example, only the domain in the graph center is bounded by its adjacent domains. The others are occupying an infinite large region of the input space. Since the number of available training samples is always limited, large portions of these domains contain no data. In consequence, it is very unlikely that the local models can approximate the target function correctly for all possible input data in such unlimited domains. A better domain model would only include regions of the input space where data samples are available and would exclude all the other regions. With such a domain model a straightforward mechanism can be realized that marks a data point that does not belong to any domain of a trained network as an outlier. The reliability of a network could then be improved

by rejecting such outliers as invalid inputs. In more detail, the properties of the center domain model and the outlier detection mechanism are discussed in Subsections 3.3.1 and 3.3.5, respectively.

In panel (b) of Figure 3.2, the same training data is captured with the hyper-elliptical domain (HED) model presented in Subsection 3.3.2. This model defines a domain as a set of hyper-ellipsoids which have a certain position, orientation and size. As noticeable in the graph, for a 2D input space this model defines ellipses that vary in their center and the direction and the diameter of their main axis. Such a domain can contain more than one ellipse to enclose all the data that is assigned to one local model (see Subsection 3.3.4 for more explanations). The line between the ellipses on the left hand side of the graph indicates such a unified domain.

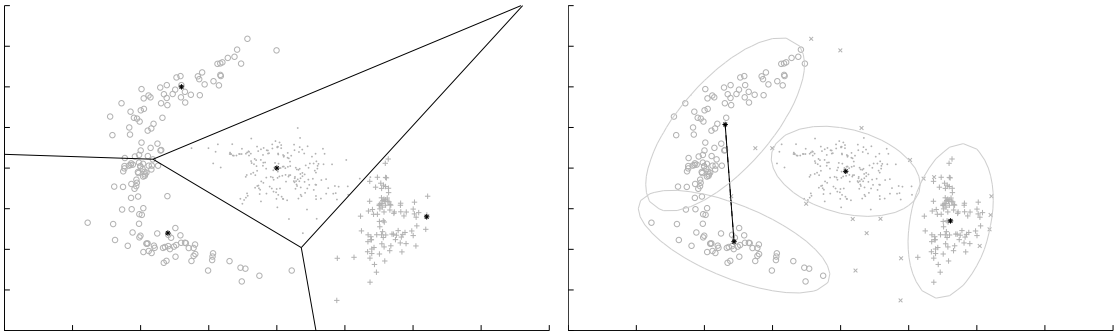
The hyper-elliptical domain model is well suited to separate the three different sample sets. The shape of the ellipses are properly adjusted to the data. The effect is that the region of the input space enclosed by one ellipse is densely sampled. So, it is reasonable to implement an outlier detection mechanism since all the regions outside of the domains do not contain data, and hence a good extrapolation quality of a local model can not be guaranteed. In contrast to the center domain model this is possible with the hyper-elliptical one: such a domain is always limited to a radial symmetrical boundary around the center of the ellipsoid. In panel (b), data samples that are rejected by this means as outliers are plotted with $\times$.

The flexibility of the HED model is also increased by the possibility to unify hyper-ellipsoids. As described in Subsection 3.3.4, one local model can be assigned to a region of the input space that is more complexly shaped than just one hyper-ellipsoid. That is the reason why the HLAM network in the example contains the optimal number of only three local models.

The drawback with the gained flexibility is the increased difficulty to estimate the parameters of the HED model. To robustly establish a hyper-elliptical domain, more data samples are needed than for a center domain. In Subsections 3.3.2, 3.3.4 and 3.6 these topics are discussed in detail.

A third division of the same training data is shown in Figure 3.3. This time, the support vector domain () model explained in Subsection 3.3.3 was used. A support vector domain is realized with an one-class support vector machine which uses data samples that are close to the boundary of a given data set. The graph also shows the so-called *activation function* of each domain. This function assigns to each value of the input space a positive scalar value that represents its "degree of membership" to a domain. E.g. the activation function of the CD model is the Euclidean distance between a data point and the domain center. The boundary of a domain is a equipotential line of the activation function. Only in the inside of a domain the activation function is higher than a certain threshold.

Compared to the center and the hyper-elliptical domain model the model is the most flexible one. The shape of such a domain can very closely be fitted to the data as visible in the Figure 3.3. Hence only three single domains can cope with the data without the need for a unification as for the hyper-elliptical domains. On the other hand, the ease to interpret an HLAM network is decreased with such complicated boundaries as it becomes harder to visualize the region of the input space that is governed by a local model.



(a) Center Domain Model (b) Hyper-Elliptical Domain Model

Figure 3.2.: The center and the hyper-elliptical domain model. The same training samples of a 2D space are divided in two different ways. The different symbols indicate that the training samples belong to different individual models. The line in panel (b) marks that the two connected ellipses are combined to one domain. Data samples that are regarded as outliers with the hyper-elliptical domain model are drawn as $\times$.

Furthermore, since a support vector domain is based on kernel functions that typically require the optimization of meta-parameters, the computational effort to establish such a domain is rather high.

3.1.2.2. On splitting up the input space

Besides the above outlined methods to model a domain, in this thesis two learning algorithms are proposed for the HLAM approach. Both define a strategy to split up the input space once in an offline and once in an online learning scenario. Both algorithms should realize a division of the input space that accomplishes the two already mentioned goals. First, the approximation performance of the local models of an HLAM network should be as good as possible. Second, the number of needed models (i.e. the number of local regions the input space was split into) should be as small as possible. These two goals are important for the performance and interpretability of an HLAM network. Unfortunately, they are strongly depending on each other.

If the number of local models is small, it is typical that each domain of a model covers a large region of the input space. Obviously, such a partitioning would be easier to understand than a division into many small parts. But the basic assumption of all modular approaches is that the target function is harder to be approximated in larger domains than in smaller ones. A large domain requires a complex machine learning technique to train an appropriate local model for it. If rather simple local models are employed in the network, the approximation performance will suffer at the local and hence at the global scale. Otherwise, with a more sophisticated machine learning technique, the computational effort will rise, and its interpretability will decrease. Hence a learning algorithm for an HLAM network should produce a partitioning of the input space that

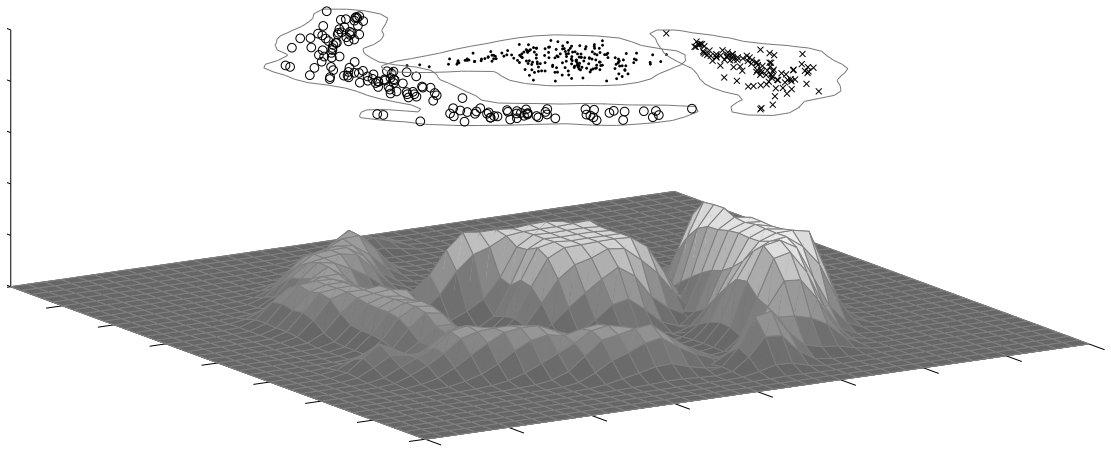


Figure 3.3.: Three Support Vector Machine domains and their activation functions for the same training data used in Figure 3.2.

is not too coarse and is not too fine.

In an offline learning scenario a set of training samples is given so that all samples can be accessed at any time by the learning algorithm. In such a case the task of splitting up the input space can be reformulated. The task is to divide the training set into subsets which have to satisfy two constraints: the samples of a subset must belong to a local region i.e. they must be covered by one single domain. Secondly, one local model must be trained with the samples to approximate the target function in such a new domain.

The first condition resembles the classical unsupervised clustering problem. A set of samples must be divided into an appropriate number of clusters. Each cluster contains those samples that are w.r.t. a given metric close to each other. This metric introduces the notion of locality to a clustering solution. So, it stands to reason to employ a clustering algorithm for the HLAM learning problem. For each cluster a new domain can be established by means that were outlined in the last section and all training samples belonging to one cluster are used to train one local model. Together with the new domain the model can be added to an HLAM network.

One minor problem with this straightforward idea is that the number of clusters is not known a priori, and the best known clustering algorithms do not determine this number automatically. But the more important problem is that a good clustering result does not imply that a good approximation of the target function can be achieved. A clustering process works only with the input samples and groups them w.r.t. the metric. In contrast to that, a supervised function approximation technique has to take input and output samples into account. This discrepancy makes it necessary to couple the clustering process with the ability of the local models to approximate the target function. Hence the two above mentioned constraints must be satisfied in a process that combines clustering and function approximation. How this can be done will be described in Subsection 3.7.1.

It will also become clear that as a byproduct an appropriated number of needed local models can be determined.

Such a coupling of clustering and function approximation needs a method to validate the performance of a local model. The clustering process must be guided by an estimate of the quality of a local model. Given a training set and a trained local model, one has to decide if the clustering process was successful, and hence the model can be added to the network. In Section 3.5 solutions to this problem will be discussed.

In an online learning scenario, the most important decision given a new training sample is whether a structural change of the network is needed or not. Either a single local model could be adapted to the new sample or a new model could be added to the network. Like in the offline learning scenario, this decision should depend on the already achieved approximation quality. So, if it seems to be sufficient to update a local model with the new data, the algorithm has to select a model for that. This decision should be based on the definitions of the domains since the models should be specialized to local regions. In the other case, when a new local model should be created, the question is where its domain should be located in the input space. The answer should depend on the positions of the already existing domains. If one knows which domains are adjacent to the new sample, one can better decide where a new domain can be inserted. Another problem is that it is not reasonable to establish a new local model and its domain with only one single training sample. For a solution a storage mechanism has to be developed that saves samples until they can be used appropriately. An online learning algorithm that tackles all these problems is proposed in Subsection 3.7.2.

Last but not least, one should note that a good partition of the input space can be achieved by the good combination of a splitting and a unifying process. The combined hyper-elliptical domain model outlined in the last subsection shows that it can be a good strategy to first produce more clusters than needed and afterward unify those which can be governed by one local model. Again the two basic constraints must be satisfied. A unified domain still has to form a local region of the input space, and the approximation performance of its local model must be sufficient. A method that unifies domains under these constraints will be proposed in Section 3.6.

3.1.3. Basic Network Definitions

The so far introduced concepts of the HLAM approach can be formalized as follows. In a supervised learning scenario, the target function $f : \mathbb{R}^n \to \mathbb{R}$ from an n-dimensional input to an one-dimensional output space is only implicitly given as a set of N training samples $T = \{(x_i, y_i)\}$ with $x_i \in \mathbb{R}^n$, $y_i \in \mathbb{R}$ and $i = 1, \ldots, N$. An HLAM network approximates the target function with the function $\hat{f} : \mathbb{R}^n \to \mathbb{R}$. That the output space is only one-dimensional is without loss of generality as one can easily combine several networks in one vector in order to realizes a mapping to a multi-dimensional output space.

An HLAM network is defined with the tuple $(g, a_1(\cdot), \ldots, a_M(\cdot), \hat{y}_1(\cdot), \cdots, \hat{y}_M(\cdot))$ where the function $g : \mathbb{R} \to [0, 1]$ defines a gating law, the functions $a_k : \mathbb{R}^v \to \mathbb{R}$ are the activation functions of the M domains with $v \leq n$ and $\hat{y}_k : \mathbb{R}^w \to \mathbb{R}$ are the local models

with $w \leq n$. Given such a tuple the target function is approximated as

$$\hat{f}(x) = \hat{y} = \sum_{k=1}^{M} g\left(a_k(x^{(D)})\right) \hat{y}_k(x^{(M)}). \tag{3.1}$$

The input space of the activation functions $a_k(\cdot)$ and of the local models $\hat{y}_k(\cdot)$ are a subspace or equal to the input space $\mathbb{R}^n$ of the target function. Their input values are denoted as $x^{(D)} \in \mathbb{R}^v$ and $x^{(M)} \in \mathbb{R}^w$, respectively. The input space $\mathbb{R}^v$ of the activation functions shall be named *domain space* since the domains of the local models are defined in this space. In contrast to that, the input space $\mathbb{R}^w$ will be referred to as *model space* in the following. This distinction allows that not all components of the input x are utilized in neither the activation functions nor in the local models. As for all known techniques presented in the last chapter, in the simplest case it is true that:

$$x = x^{(D)} = x^{(M)}. \tag{3.2}$$

This explicit discrimination between the domain and the model space is a specific feature of the HLAM approach and will be motivated in the next subsection.

The gating law $g(\cdot)$ is a function that weights each local model w.r.t. its activation. A gating law defines how the outputs of the local models for a given input are combined in order to compute the overall output $\hat{y}$ of an HLAM network. Different gating laws that are suitable for different purposes will be given in Section 3.2.

The activation functions $a_k(\cdot)$ are essentially representing the domains of the local models. An activation function has to be defined for the domain space and should express with a real value how much an input value "activates" a local model. Figure 3.3 shows an example of activation functions. In Section 3.3 different domain models will be proposed along with their different activation function definitions.

Finally, the local models $\hat{y}_k(\cdot)$ are approximating the target function. As it should have become clear, the overall concept of the HLAM approach demands that the local models are specialized to their domain. What type of parametric model is used for $\hat{y}_k(\cdot)$ is a free parameter of the approach. Even different types of models could be used in the same network. Some suggestions for that will be discussed in Section 3.4. This freedom of choice is the base for the hierarchical extension of the HLAM approach: an HLAM network itself can be used as a local model. The benefits of this idea will be discussed in the following subsection.

3.1.4. Advanced Features

In the last subsection the distinction between the domain and the model space was introduced. Both are subspaces of the input space of the target function. The domain space contains the domains of the local models. While the model space is the input space of the local models. From this space the mapping to the output space of the target function is realized. This distinction is motivated by the fact that two different tasks have to be solved by an HLAM network in order to approximate the target function. On one hand, certain local models have to be selected that should define the output of

the whole network. On the other hand, the selected models have to compute their own output. The first task is solved w.r.t. the domains of the local models. While the second one is solely based on input data of the model space.

So, for a specific application one can select appropriate components of the target function's input space for the domain and the model space. By this means one can make use of available knowledge about the data that is processed by an HLAM network. Generally speaking, only those components that are amenable to form certain classes of data should be used for the domain space because the domains group this input data together that is processed by one local model. Especially in the presence of inhomogeneous information sources (i.e. where different continuous and discrete input data have to be processed), a distinction between the domain and model space stands to reason. In Chapter 7 networks are presented that demonstrate this feature of the HLAM approach.

Taking a more abstract view, the domain space defines the context information of the input data. On the other hand, the components of the model space convey the fundamental information that has to be transformed to the output. By means of the contextual information, input values of the model space are grouped to certain classes. These classes of data are processed by the local models. Hence the same fundamental input values can lead to different output values. The result depends on the context information.

Another aspect of this projection of the HLAM network is that the automatic process of establishing domains can be interpreted as finding different context classes. Hence given a trained HLAM network, one can analyze how the fundamental part of the input is grouped by the contextual one. One has to examine the domains and will find out which data points belong together as these are situated in the same local region of the domain space.

So, one should select those input components for the domain space that can serve as context information. Along this way, a dimension reduction will also be realized since the original input space of the target function is split. This can considerably help to establish robust parameter estimation for both the domains and the local models. Otherwise problems can occur due to high dimensional input data, especially if only a small number of training samples are available.

The separation between the domain and the model space is a distinct feature of the HLAM approach. Although most of the state of the art techniques presented in the last chapter can be extended to realize the same distinction it has not been proposed in the literature, yet. The different authors do not discuss the two separated tasks of selecting a local model and computing its output.

Another advanced feature of the HLAM approach is that different possibilities exist to built up hierarchies of networks. The most straightforward and in the last section already mentioned way is to use the HLAM technique itself to train the local models. Whenever a domain was established and a local model has to be fitted to the domain's data samples a new instance of the HLAM learning algorithm can be evoked to produce a new local model. The result would be a hierarchy of networks. At every level of the hierarchy the domain of a local model would be divided with a finer resolution by the HLAM network of the lower level. The full hierarchy would contain domains which are

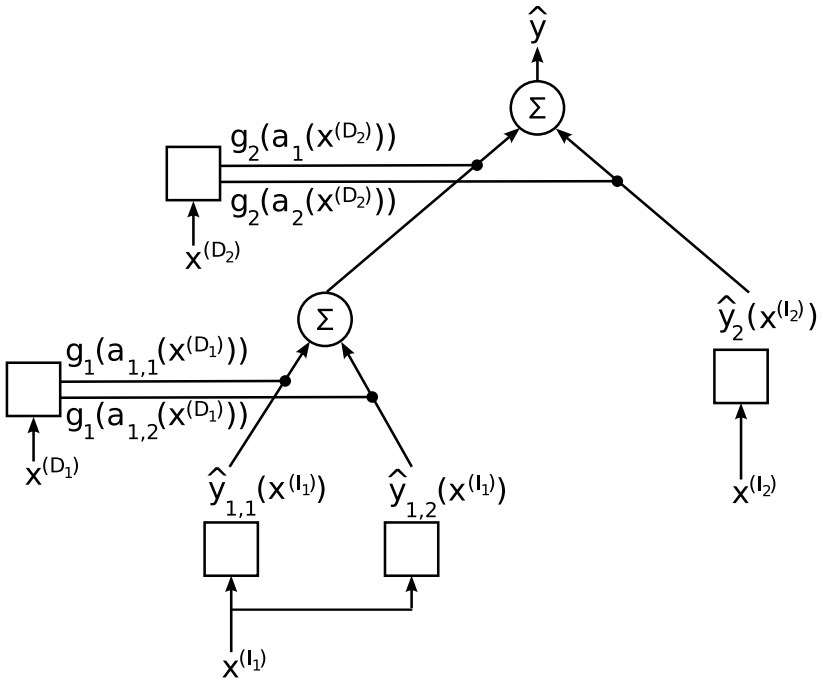


Figure 3.4.: A two-level HLAM network. Three local models ($\hat{y}_{1,1}(\cdot), \hat{y}_{1,2}(\cdot)$ and $\hat{y}_2(\cdot)$) with two different model spaces (I_1 and I_2) produce the output $\hat{y}$. They are combined by two gating laws ($g_1(\cdot)$ and $g_2(\cdot)$) which work on data from two domain space (D_1 and D_2).

separated in subdomains which again are separated in subdomains and so on. Such a network of networks can be analyzed so that with arbitrary depth groups of subgroups of data could be distinguished. Of course it depends highly on the data to which extend such a hierarchy of networks is reasonable.

Another way to built up a hierarchy is to couple networks that are trained with different data sets. In a first step one would collect different training sets and manually train local models for each of these. Afterwards these can be combined to an HLAM network by establishing domains based on the different training sets. This can be done by the same means as if the training set was split into subsets by an automatic process. It is important to note that the model spaces of the combined local models can be different. They are coupled by their domain and output space. These have necessarily to be equal. Figure 3.4 shows an example HLAM network that has two levels and makes use of the full flexibility of different domain and model spaces and gating laws.

The separately trained local models can be regarded as modules that can be plugged together. They are specialized to subproblems that are expressed by the different training sets that are selected manually. In this way application specific knowledge can be introduced into a hierarchy. That can ease the process of solving of a given learning prob-

lem. For example, a hierarchical classification scheme for object categorization could be realized in the following way: on the lowest level two local models could be trained where one separates oranges from apples and another distinguishes between cars and houses. The model space of the former may contain color features of the object's appearances, while the latter works on shape features. Their common domain space can supply more abstract knowledge that may allow a distinction between natural and artificial objects. On a next hierarchical level this network could be combined with a network that was trained to recognize objects that are given with an auditive features description. Those subnetworks could be coupled by a domain space that reflects the temporal context in which an user showed an image or made an utterance to the system. In theory this hierarchy could be extended arbitrarily in this way. The user can integrate more information source and can decide at which level these are most appropriate to distinguish more classes of objects.

The example shows that the HLAM approach makes it possible to realize multi-level heterogeneous hierarchies. Trained local models can be combined as specialized modules to a complex network. One can mix various machine learning techniques to train these modules. This allows that appropriate techniques are employed for differently challenging data. Due to the distinction between the model and domain space various information sources can be distributed over a hierarchy. The diverse domain and model spaces at the different levels of a hierarchy serve as a multiple cue integration.

In summary, the HLAM approach offers a coherent framework to create a modular solution to a learning problem at hand. It can be seen as a construction kit where several options exists to customize a hierarchical network of locally arranged models. The next sections will propose a number of such options that are concerned with certain aspects of an HLAM network (e.g. different domain models, gating laws or learning algorithms).

3.2. Gating Laws

As already outlined in Subsection 3.1.3, the gating law of an HLAM defines how the output of all the local models is combined to the output of the whole network. With the gating law it will be decided to which extend the different local models cooperate i.e. their output is mixed. The two most extreme cases are that either all models equally contribute to the output of the network or that only one model determines it. A compromise would be a gating law where only a small number of local models dominate the output.

According to (3.1), a gating law is a function $g : \mathbb{R} \rightarrow [0, 1]$ that, given a certain input $x^{(D)}$, uses the activation value of a domain to compute a weight for the corresponding local model.

In the following two gating laws will be proposed. The first is the extreme case where only one local model determines the output of the whole network. While the second one defines a mixture of models where some of them dominate the overall output. The other extreme case that assigns equal weights to all models was not pursued in this thesis as it is not consistent with the idea of the HLAM approach: the output of any local model would be used regardless to which extend it was activated i.e. regardless to their domains.

3.2.1. The Exclusive Gating Law

Exactly one local model determines the output of an HLAM if the exclusive gating (EG) law is applied. It is defined as:

$$g_{\mathrm{EG}}\left(a_k(x^{(D)})\right) = \begin{cases} 1 & : \quad k = \arg_i \max\ a_i(x^{(D)}) \\ 0 & : \quad \text{else} \end{cases} . \tag{3.3}$$

So, the maximum weight of 1 is only assigned to the model that has the highest activation value for a given input. In simple words, it is a "winner takes all" rule. All the other models of an HLAM will have a weight of zero so that equation (3.1) can be simplified to:

$$\hat{y}(x) = \hat{y}_k(x^{(M)}) \text{ with } k = \arg_i \max a_i(x^{(D)}). \tag{3.4}$$

An implementation of the EG law should take two special cases into account. In one case no local model could be activated i.e. $\forall k: \ a_k(x^{(D)}) = 0$. Then the output of the network would be zero which should be marked as the absence of any activation (for more on this topic see Subsection 3.3.5). Otherwise this output could not be distinguished from a valid output of 0 of a local model. In the other special case, the activation value of a number of local models is equal. In practice, this should not happen since the activation values are given as floating point numbers which are very rarely equal. But in theory, the definition of (3.3) becomes ambiguous. Without additional knowledge the question which of the activated models should compute the output is undecidable. Hence an implementation can select randomly one model.

The EG law simplifies the interpretation of a domain model. The applied "winner takes all" principle draws a sharp boundary between two adjacent domains. Either the output of a local model is taken as the network's output or it is not. So, the intersection of the activation function of two adjacent domains defines the boundary between them. This implies that a certain domain has to be analyzed w.r.t. its surrounding domains (see Subsection 3.3.1 and Subsection 3.3.2 for examples of this dependency).

The sharp boundaries between domains have another implication. They typically result in discontinuities of the approximation of the target function. Such jumps are visible in the example function shown in Figure 3.1. This property has be kept in mind if an HLAM is applied in an application. Obviously, in cases where it is assumed that the target function is continuous, an approximation with the EG law may create difficulties. Then two questions are important: is the HLAM used repeatedly in order to transverse the input space in a continuous manner? And, does the resulting jumps in the network's output really cause problems? An example for that could be the case where real valued output of an HLAM is used to control a plant continuously over time. Still then, the questions remain how tolerant the controlled plant is against quick changes in their control parameters and how large the jumps are. The latter depends on the difference of the local models that can be controlled by the parameters of the learning algorithm. Otherwise an HLAM with the EG law should not cause more practical problems than other approximation techniques with smooth output functions.

On the other hand, the EG law offers advantages if the target function has a number of discontinuities or is discrete by definition. In the first case techniques as RBF networks would produce inadequate oscillations around jumps. The latter will always be true if a classification problem has to be solved. The different classes will be coded as integers, and a smooth transition between them does not make sense. In such a case the EG law would be very appropriated since it draws sharp boundaries between the domains that would model the different classes. Note that none of the state of the art methods presented in the last chapter offers this feature.

3.2.2. The Mixing Gating Law

The mixing gating (MG) law defines the activation weight of a local model relative to the activation of all the other models. It is given with:

$$g_{\mathrm{MG}}\left(a_k(x^{(D)})\right) = \frac{a_k(x^{(D)})}{\sum_{i=1}^{M} a_i(x^{(D)})}. \tag{3.5}$$

This definition assumes that the activation functions are always positive. If this is not the case, any transformation that shifts the activation functions to positive values could solve this problem. E.g. one could use the $\exp(\cdot)$ function for a transformation.

The mixing gating law resembles the "soft-max" function (2.10) on page 16 used in the Local Linear Map approach and offers the same benefit: The resulting output function $\hat{y}(\cdot)$ is a continuous function since the output of the local models are mixed w.r.t. their continuous activation functions. That realizes the smooth transition between different models that can not be accomplished with the EG law. On the other hand, discontinuities of the target function can not be reproduced appropriately.

Compared to the EG law, it becomes harder to analyze an HLAM when the MG law is applied. As discussed on page 12, with (3.5) the same effects of normalization occur and are distorting established domains. That can considerably complicate the question which local model dominates the output of the whole HLAM. A similar problem concerning this question results from activation functions that have more than one local maximum. E.g. the support vector domain may have different local maxima (cf. Figure 3.6). Hence the output of a local model will be weighted differently within its own domain. Still, it depends strongly on the actual HLAM how prominent this effect may be.

3.3. Domain Models

In the following three different domain models are proposed. Their formal definition (i.e. the parametrization of their activation function) will be given. During the training of an HLAM such domains are established by different parameter estimation methods that will also be explained. Of course these methods are specific for the different domain models. They have in common that they work on the domain space components of the training data.

Algorithm 3.1: Pseudo code of the learning algorithm for the center domain model.

Function `EstablishCenterDomain`
Input : $T = \{x_i^{(D)}\}$
Output: $\Phi_{\text{CD}} = p$
begin
$p \leftarrow \frac{1}{m} \sum_{i=1}^{m} x^{(D)}$
end

To systematize the different domain models, the following nomenclature is used throughout this text. The tuple Φ with an appropriate subscript denotes the parameters of a specific domain model. These parameters are estimated given a set $T = \{x_i^{(D)}\}$ of m training samples. In the online learning algorithm (s. Subsection 3.7.2) the tuple Φ^t is also indexed with the step number t.

As pointed out in Subsection 3.1.2.1, the domain models have great impact on the performance and interpretability of an HLAM. So, the properties of the different domain models will also be discussed. Note that for these argumentation one has to keep a specific gating law in mind since for example the shape of a domain will be distorted if the MG gating law is applied. If not otherwise stated, a domain model will be discussed under the assumption that the exclusive gating law defined in Subsection 3.2.1 is used. It allows the clearest interpretation of a domain model.

Given the three domain models a formal definition will be given in Subsection 3.3.4 how such domains can be unified. This whole section will be concluded with remarks how outliers in the data can be rejected and about implications of the different gating laws to the domain models.

3.3.1. The Center Domain Model

The center domain (CD) model consists of a prototype vector $p \in \mathbb{R}^v$. Its activation function is defined as the negative Euclidean distance between that prototype p and the input vector $x^{(D)} \in \mathbb{R}^v$:

$$a_{\text{CD}}(x^{(D)}) = -\|p - x^{(D)}\|. \tag{3.6}$$

Accordingly, the domain parameters are defined with:

$$\Phi_{\text{CD}} = p. \tag{3.7}$$

The learning algorithm for the CD model is very simple and given as pseudo code in Algorithm 3.1. The prototype p is set to the arithmetic mean of all the m training samples belonging to a domain.

The CD model offers the great advantage of simplicity. With the prototype p only one parameters has to be determined and that can very simply be estimated by the stated

method. Furthermore the interpretation of the CD model is straightforward as it results together with the exclusive gating law in a Voronoi tessellation of the input space. Basically, this is a nearest neighbor decision that draws linear boundaries between different domain (for an example see panel (a) of Figure 3.2). This has a number of drawbacks: the nearest neighbor principle does not necessarily imply that an input assigned to a certain domain is near to its prototype and hence to the training data that was used to establish the domain. The result is that a local model can be assigned to an input that is totally different to the model's training data. This will happen just because no other prototype was closer to the input data. As already discussed on page 24 the problem is that an instance of a CD model can be unlimited large but the number of training data will always be limited. Hence the validity of a local model can not be guaranteed for its whole unlimited domain.

Another disadvantage is that the shape of a domain is quite strongly restricted. Only a convex region of the domain space that is bounded by hyper-planes can be modeled with a CD model. Somehow rounded shapes can only be approximated by increasing the number of domains in an HLAM. More domains would mean that more hyper-planes will divide the domain space in a finer resolution, but would also mean that the computational effort will increase.

With the CD model adjacent domains are highly interdependent. The boundary between two domains is the hyper-plane that is perpendicular to the connection of the domains' two prototypes and lays halfway between these two. The latter is the reason why a boundary will move if the position of a domain's prototype is shifted. Moving one domain will always increase or decrease the size of all surrounding domains.

3.3.2. The Hyper-Elliptical Domain Model

As the name indicates, with the hyper-elliptical domain (HED) model the shape of a domain forms a hyper-ellipsoid in the domain space. As the base for the HED model, a hyper-ellipsoid is defined with the function $d_{\mathrm{HED}} : \mathbb{R}^v \to [0, 1]$ as:

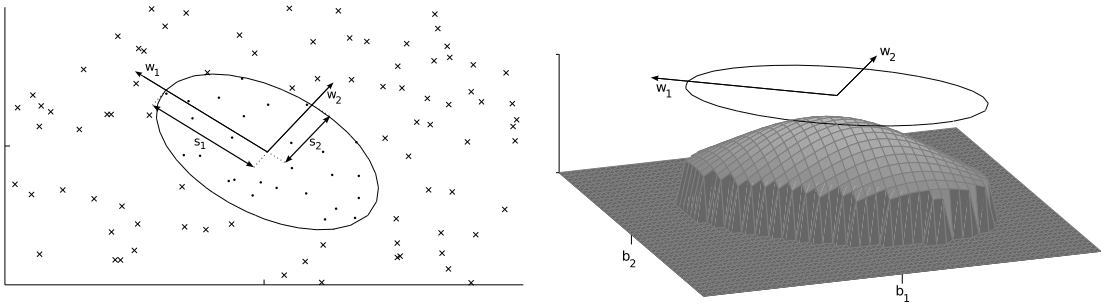
$$d_{\mathrm{HED}}(x^{(D)}) = \exp\left[-\sum_{i=1}^{v}\left(\frac{1}{s_i}(x^{(D)} - p)w_i\right)^2\right]. \tag{3.8}$$

This is a special parametrization of a Gaussian function that allows a clear interpretation: The v-dimensional hyper-ellipsoid is centered in the domain space at position $p \in \mathbb{R}^v$. The v axes of the hyper-ellipsoid are defined with $w_1, \ldots, w_v$ ($w_i \in \mathbb{R}^v$) and stretched with the scalar values $s_1, \ldots, s_v$ where $\forall i : s_i \in \mathbb{R} \geq 0$. So, the function $d_{\mathrm{HED}}(\cdot)$ has it's maximum at p and decreases along the hyper-ellipsoid's axes w_i to zero.

Since the scalar values s_i may be zero one special case of (3.8) has to be treated: if one scalar s_i is equal zero and the i^{th} components of the position p and the input $x^{(D)}$ are not equal, then the value of the function $d_{\mathrm{HED}}(\cdot)$ is set to zero.

Given $d_{\mathrm{HED}}(\cdot)$ the activation function of the HED model is defined as:

$$a_{\mathrm{HED}}(x^{(D)}) = \begin{cases} d_{\mathrm{HED}}(x^{(D)}) & : d_{\mathrm{HED}}(x^{(D)}) \geq \gamma \\ 0 & : \text{else} \end{cases}, \tag{3.9}$$



(a) (b)

Figure 3.5.: Examples of a hyper-elliptical domain. Panel (a) visualizes the domain parameters for a 2D domain space and some example data (dots belong to the domain, crosses do not). In panel (b) the activation function for the same domain is shown. Note that the activation functions drops abruptly to zero outside of the ellipse.

where the threshold γ is a scalar parameter. So, the activation value of a domain for a given input $x^{(D)}$ is equal to the value $d_{\text{HED}}(x^{(D)})$ if this value is equal or greater than the threshold γ. Otherwise the activation is zero. By means of the threshold parameter γ it is determined whether the input $x^{(D)}$ is enclosed by the domain's hyper-ellipsoid or not. It defines a sharp boundary that states whether a sample belongs to or does not belong to a domain. Figure 3.5 shows an example of a HED model for a 2D domain space.

A hyper-elliptical domain is fully specified with the parameter tuple:

$$\Phi_{\text{HED}} = (p, s_1, \ldots, s_v, w_1, \ldots, w_v, \gamma). \tag{3.10}$$

The basic idea to estimate these parameters is to use *principle component analysis* (PCA). A compact description of this statistical standard method can be found in [8]. PCA is applied onto the data that belongs to one domain. The resulting principle components define the axes of its hyper-ellipsoid. The diameter of each axis reflects the variance that the corresponding component captures. The hyper-ellipsoid are centered on the data. The threshold parameter is chosen so that the data sample maximal apart from the domain's center is still included by the domain. By this means the smallest hyper-ellipsoid can be constructed that encloses all the given data samples.

The pseudo code for the implementation of this idea is shown in Algorithm 3.2. It works as follows. The center position p of the hyper-ellipsoid is set to the mean value of the training data. Then a PCA is performed i.e. the covariance matrix of the training set is computed. The axes parameters w_i are set to the eigenvectors of this matrix as these represent the principle components. Given these axes, the scaling factors s_i are determined with the following geometrical interpretation of the scalar product of two vectors: they are set to the maximal value of the projection of the training samples onto the axes. Finally, the threshold parameter γ is computed using the already established parameters: γ is set to the minimum activation value of the training samples. That ensures that all the training samples are enclosed by the new established domain.

Algorithm 3.2: Pseudo code of the learning algorithm for the HED model.

Function `EstablishHyperEllipticalDomain`
Input : $T = \{x_i^{(D)}\}$
Output: $\Phi_{\text{HED}} = (p, s_1, \ldots, s_v, w_1, \ldots, w_v, \gamma)$
begin
  $p \leftarrow \frac{1}{m} \sum_{i=1}^{m} x^{(D)}$
  forall $j \in \{1, \ldots, v\}$ **do**
    $w_j \leftarrow j$-th principle component of set T
    $s_j \leftarrow \max_i \|(x_i^{(D)} - p)^T w_j\|$
  $\gamma \leftarrow \min_i d_{\text{HED}}(x_i^{(D)})$
end

The special feature of the HED model is the sharp boundary between the inside and the outside of the hyper-ellipsoid. It simplifies the decision if or if not a sample belongs to a domain. This question can be determined independently from other domains of an HLAM. In contrast to the CD model the activation value of one domain has not to be compared to other activation values due to its threshold parameter γ. Hence the domain boundary is fixed. It ensures that the domain is restricted to the local region in the domain space where data samples were actually available. A hyper-elliptical domain is always bounded. That should increase the reliability of a local model that is trained with a limited number of samples from a restricted local region.

Since the HED model defines such a fixed boundary the interpretation of an HLAM is simplified. Each established domain can be examined individually. Hence independent insights about the local models of an HLAM can be gained. The data one local model is assigned to can be described in terms of their mean position in the domain space and their variance along different axes. This interpretation is directly given with a domain's parameters Φ_{HED}. Since these are established with a PCA one can find the most important direction in which the data of a local model is spread out in the domain space. Those axes of the hyper-ellipsoid that have comparatively small scaling factors can be neglected in an analysis of a domain.

One limitation to the local analysis of a hyper-elliptical domain exists: the independence of different domains of an HLAM does not only depend on the HED model. It also relies on the process that generates the training data to establish the different domains. As visible in Figure 3.2 hyper-ellipsoids can overlap each other. This might happen in cases where data that is supposed to be approximated by different local models is located closely together in the domain space. With the exclusive gating law applied in such cases the shape of a domain would become an intersection of different hyper-ellipsoids. This would complicate the interpretation of established domains.

Another drawback of the HED model is that the estimation of its parameters is not as robust as for the CD model. The determination of the hyper-ellipsoid's axes w_i and hence

of the scaling parameters s_i is sensible to noise in the data. This can result in problems in cases where only a small number of training samples is available. Furthermore the use of PCA introduces an assumption that is common among other approaches [64, 68, 99]. The hypothesis is that the data belonging to one domain is generated by a Gaussian random variable. If this is not the case, the established hyper-ellipsoid will not adequately enclose the training data.

3.3.3. The Support Vector Domain Model

The support vector domain (SVD) model is an one-class support vector machine [89, 100] that is trained to classify if a sample belongs or does not belong to a domain. Analogous to the HED model, a function $d_{\mathrm{SVD}} : \mathbb{R}^v \to \mathbb{R}$ is defined as:

$$d_{\mathrm{SVD}}(x^{(D)}) = \sum_{i=1}^{S} \alpha_i K(s_i, x^{(D)}), \tag{3.11}$$

where $S \in \mathbb{N}$ is the number of the support vectors $s_i \in \mathbb{R}^v$ with their weights $\alpha_i \in \mathbb{R}$ and $K : \mathbb{R}^v \times \mathbb{R}^v \to \mathbb{R}$ is a kernel function. Typical choice for K is the radial basis kernel:

$$K(x, y) = \exp(-\frac{1}{\sigma}\|x - y\|^2). \tag{3.12}$$

With $d_{\mathrm{SVD}}(\cdot)$, the activation function of the model is given as:

$$a_{\mathrm{SVD}}(x^{(D)}) = \begin{cases} d_{\mathrm{SVD}}(x^{(D)}) \colon & d_{\mathrm{SVD}}(x^{(D)}) \geq \gamma \\ -\infty & : \text{else} \end{cases}, \tag{3.13}$$

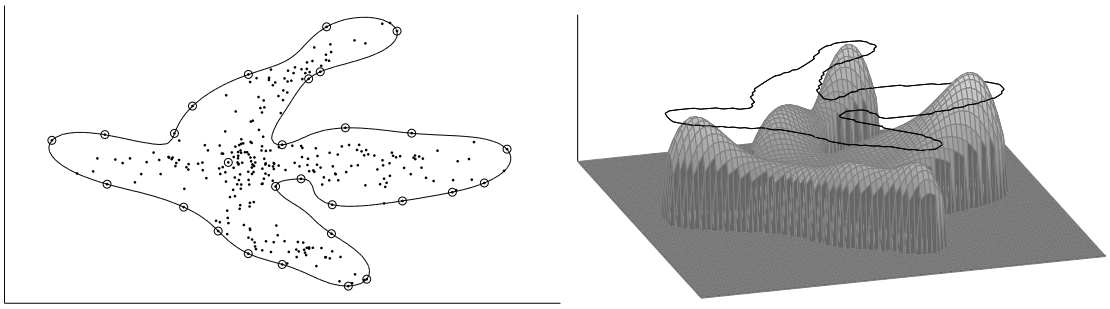
where the threshold γ is a scalar parameter. Like for the HED model, the activation value is equal to the value of (3.11) if the threshold is met or exceeded. Again a sharp boundary is drawn around the domain which strictly separates the domain's inside from its outside. In contrast to the definition of $a_{\mathrm{HED}}(\cdot)$, the outside of SVD domain is marked with the value minus infinity instead with zero. That is necessary since – depending on the chosen kernel function – $d_{SVD}(\cdot)$ may be negative. Figure 3.6 shows an example of a support vector domain.

For a support vector domain the following parameters have to be estimated:

$$\Phi_{\mathrm{SVD}} = (s_1, \ldots, s_S, \alpha_1, \ldots, \alpha_S, \gamma).$$

For a given data set, the support vectors (SV) and their weights are determined with the standard method for one-class support vector machines as defined in [89, 100].[1] Note that additional parameters for the employed learning algorithm and the chosen kernel function have to be specified. In the following these (e.g. the regulation parameter ν for the algorithm or the kernel width σ for a radial basis kernel) are referred to as meta-parameters of the SVD model. They have to be adequate and hence have to be manually optimized for specific training data (for a discussion see further below). The threshold γ

[1] The actual implementation uses the free toolbox [21].



(a) (b)

Figure 3.6.: Example of a support vector domain. Panel (a) shows a set of data samples and the domain boundary realized with a radial basis SVD. These samples that are support vectors are marked with circles. Panel (b) shows the corresponding activation function.

is set to the minimum value of the function $d_{\mathrm{SVD}}(\cdot)$ of all support vectors s_i. Like for the HED model this establishes a domain boundary that encloses all the training data. The only difference is that not the minimum of all samples but just of the support vectors has to be found because these are supposed to be located at the boundary. The complete learning algorithm is stated in Algorithm 3.3.

The SVD model possesses the same properties as the HED w.r.t. its fixed boundary: an established domain is restricted to the local region of the domain space where actual data was available. Due to the threshold γ the question about the membership to a certain domain can be decided definitely and without comparisons to other domains. Still different domains may overlap each other so that the analysis about a realized partitioning of the domain space may remain not easy.

Of the three proposed domain models the interpretation of a support vector domain is the most complicated. The set of support vectors s_i has to be examined to understand to what local region a local model is assigned to. As explained in [100], they define a convex hull around the training data which is the boundary of a domain. The problem is that the convex hull is not given in the domain but in the feature space that is induced by the used kernel function. Hence a domain's boundary is not the linear connection between adjacent support vectors but can follow a complicated non-linear function.

On the other hand, this property of a boundary defined with support vectors introduces a great flexibility to the SVD model. Compared to the HED or CD model a support vector domain can be fitted more tightly to a given set of training data. With the more flexible SVD boundary the volume that a domain encloses can be reduced. One resulting advantage is that different domains of an HLAM will have less overlap hence the interpretation of the network should be simplified. The other is that more data samples which may be widespread in the domain space can be adequately enclosed by one domain and hence approximated by one local mode. Since in such a case more data is available to train a local model its approximation performance should improve. Furthermore, with larger domains the resulting HLAM becomes smaller. If more data can be assigned to

Algorithm 3.3: Pseudo code of the learning algorithm for the SVD model.

Function `EstablishSupportVectorDomain`
Input : $T = \{x_i^{(D)}\}$
Output: $\Phi_{\text{SVD}} = (s_1, \ldots, s_S, \alpha_1, \ldots, \alpha_S, \gamma)$
begin
  Solve for s_j and α_j:

$$\min_{\alpha_j} \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j K(x_i^{(D)}, x_j^{(D)}) \quad \text{subject to} \quad 0 \leq \alpha_j \leq \frac{1}{\nu}, \sum_j \alpha_j = 1$$

  $\gamma \leftarrow \min_j d_{\text{SVD}}(s_j)$
end

the single domains, their number will be reduced. That is beneficial for an interpretation of an HLAM.

The flexibility of the SVD model is attained with its comparative complexity of its parameters and their estimation. The number of support vectors is not fixed. In the worst case all training samples would be SVs. The number of SVs essentially depends on the meta-parameters which include a choice of a kernel function. Figure 3.7 shows four support vector domains trained for the same training data but with different meta-parameter values. For all four domains a radial basis function (cf. (3.12)) was used as kernel function. Only the kernel width σ was changed. This figure makes obvious that the established domain boundary and the set of SVs change greatly. With decreasing kernel width the boundary becomes more bent and encloses the data samples tighter. Simultaneously the number of support vectors increases. Apparently, one has to choose carefully the meta-parameters in order to prevent a too large or a too tight result. In the first case the realized solution would be too general and would include regions to a domain that do not belong to the associated local model. On the other hand the domain would be overfitted to the data so that the approximation performance suffers, too.

In Figure 3.6 one can also see that a support vector domain may contain holes or in more extreme cases may become a set of single parts. This is quite different to the two already proposed domain models which in any case form a connected and convex local region in the domain space. Whether such holes degrade the performance of an HLAM can not be ascertained in a general discussion. It also does not contradict the main idea of the HLAM approach that a local model should be assigned to a local region of the domain space. This locality constraint is fulfilled since the partitioning processes proposed in Section 3.7 ensure that the training data for one domain stem from a local region. The possibility of holes in a support vector domain has only be kept in mind if an HLAM should be interpreted.

So, the choice of the meta-parameters is important as it affects the performance and interpretability of an HLAM. The difficulty is that the parameter estimation process of a single support vector domain is embedded in the learning algorithm of the whole network. The choice can only be grounded on general information about a trained HLAM: the approximation performance of the HLAM, the number of established domains and the

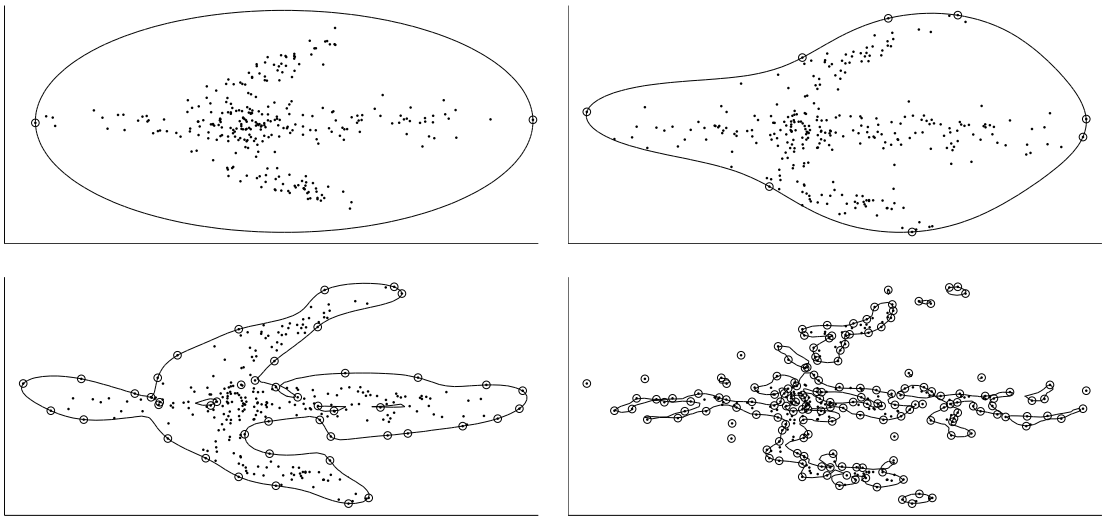


Figure 3.7.: Examples of radial basis support vector domains with different kernel widths. Support vectors are marked with circles. From the top-left to the right-bottom panel the width is decreased.

number of support vectors of each domain (in relation to the number of the used training data). Since the first two results depend also on parameters of the partitioning process, no direct feedback is available to adjust the meta-parameters of the SVD model. These difficulties in estimating and analyzing the parameters of the SVD model is the price paid for its flexibility and independence of any assumption about the data distribution.

3.3.4. Unified Domains

The proposed domain model can be extended with a quite simple but powerful idea: adjacent domains of a trained HLAM can be unified. The result is that only one local model is assigned to all the local regions of the previously single domains. That increases the flexibility how a domain of a local model can be shaped. It improves the approximation performance of the local model as more data samples are available to train it more robustly. Additionally, the unification of domains decreases the number of local models which is beneficial for an analysis of an HLAM. In the rest of this section, two ways to unify a set of domains are described. Such a method will be denoted as `UnifyDomain` in algorithms in this thesis. The more complex question which of the domains should be combined is discussed in Section 3.6.

One can choose from two options how different domains should be unified: either a new domain is trained that should enclose all the previously single domains or the established domains will not be modified and only be associated together. The first option can be implemented in a straightforward manner. The learning method for a chosen domain model is performed on the unified training sets of the single domains. Likewise a new local model is trained on the same training set. The gained domain and local model

simply replaces the set of domains and models in the HLAM. Formally, it can not be distinguished from a not unified domain.

The second option unifies a set of r domains which are given as $\Phi_1, \ldots, \Phi_r$ with the new activation function:

$$a_{\text{UNI}}(x^{(D)}) = \max_{1 \leq i \leq r} a_{\Phi_i}(x^{(D)}), \tag{3.14}$$

where $a_{\Phi_i}(\cdot)$ is the activation function of the i^{th} domain Φ_i. Accordingly, the parameters of the unified domain are defined as the tuple:

$$\Phi_{\text{UNI}} = (\Phi_1, \ldots, \Phi_r). \tag{3.15}$$

So, with (3.14) and (3.15) single domains are unified by an association that outputs the maximal activation value of the single domains as the activation value for the unified domain. Note that this definition is general enough to unify different types of domain models. To complete the unification process, a local model has to be trained and assigned to the new unified domain. To do so, the training sets of the single domains are simply unified and processed by the machine learning technique for the chosen type of local model.

If one compares these two options for unification against the background of the three proposed domain models, one should note the following. In most cases, option one should fit best to the support vector domain model. This model offers the greatest flexibility to enclose data that was previously distributed over several domains. The other two models will not adequately capture (i.e. tightly enclosed) data that is too scattered. On the other hand, these methods should be well suited for the unification by association. Especially the hyper-elliptical domain model is enhanced. Its shape gains a fruitful flexibility as with a set of hyper-ellipsoid also convex domains can be modeled. Still the good interpretability and the ease of parameter estimation of each single hyper-ellipsoid is preserved.

3.3.5. Outlier Rejection

An outlier should be understood as with input values that are atypical for a specific application. Such values may stem from a noisy sensor or may be the result of a failure of the method that provides the input. If such data samples are passed to a trained standard machine learning technique, output values will be computed that are not trustworthy. For example a linear model or multi-layer perceptron would always calculate some output regardless how invalid the input was. Without notification, the error of the input acquisition method would be propagated to the consecutive processing steps of a system. If this can cause major problems (e.g. in a controlling task), it would be very helpful to have a mechanism that automatically detects such outliers in an early stage of a processing chain. The HED and SVD models offer such a feature. It increases the reliability of a trained HLAM.

Due to their fixed and sharp boundary (induced by the threshold parameter γ), the question whether the data sample belongs to a specific domain of an HLAM can be definitely decided. If the data sample does not belong to any domain of a trained network,

it can be marked as an outlier. With (3.1) on page 29, a data sample $x^{(D)}$ is an outlier if, and only if, for all $k = 1, \ldots, M$ the activation functions $a_k(x^{(D)})$ are equal zero for the HED model or minus infinity for the SVD model, respectively. In such a case the implementation of the HLAM approach should throw an exception or use another error mechanism to notify the consecutive processing modules about the outlier detection. Simply sending the zero as a result of (3.1) can not be distinguished from a zero that may be the valid output of a local model.

This mechanism works if the training data does not contain outliers. The data could be affected by noise, but only up to a level that is acceptable. So, one has to select which input data should be regarded as valid. If this is not possible or too time demanding, the domains can be trained in a more conservative way: instead of setting the threshold parameters γ to the minimum of the activation values of the training set it could be set to a larger value. This will decrease the size of a hyper-ellipsoid or a support vector domain so that more data samples will be rejected as outliers.

There is a number of reasonable possibilities to choose a higher value for γ. The threshold could be set to the mean or the median of the training datas' activation values. That is very simple to implement but quite restrictive. A more flexible solution would be to determine a threshold that is below the activation value of a certain percentage of the training data. One could deliberately choose that, say, 95 % or just 80 % of the training data should be enclosed by a domain. One could ground the percentage level based on the available knowledge about the processed data.

Finally note, that this outlier detection can also be interpreted as a novelty detection. It depends on the view point i.e. on the task at hand if a data sample should be regarded as an invalid input or if the same sample is taken as a new piece of information. In the first case the sample should not be processed as it may cause harmful effects. In the latter it could be used to extend a trained HLAM. This idea is utilized in the online learning algorithm proposed in Subsection 3.7.2.

3.3.6. Final Remarks on the HED and SVD Models

The HED and SVD models feature a sharp boundary that separates the inside from the outside of a domain. This is advantageous for outlier detection and for the online learning algorithm. But it does not fit to the mixing gating law (cf. Subsection 3.2.2) and to an application that does not need an outlier rejection. In the first case, the sharp boundaries create discontinuities in the activation function. These annihilate the desired effect of the mixing gating law that should allow a smooth transition between different local models. On the other hand, an application may not be able to handle rejected outliers. It may need at least the best guess a trained HLAM can offer. But with the definitions of (3.9) and (3.13) the output is always zero if the input does not belong to any domain at all.

Both problems can be solved by setting the activation function of the HED and SVD models equal to the functions $d_{\mathrm{HED}}(\cdot)$ and $d_{\mathrm{SVD}}(\cdot)$, respectively. These functions defined in (3.8) and (3.11) are continuous and thus match the idea of the mixing gating law. This alternative definition is also beneficial if an outlier rejection is not needed. If a

certain input does not belong to any domain, this local model will dominate the output of the HLAM which domain is closest to the input. That is grounded in the fact that $d_{\mathrm{HED}}(\cdot)$ and $d_{\mathrm{SVD}}(\cdot)$ have their global maximum inside of their domains. This kind of nearest neighbor selection suits well the basic idea of the HLAM approach that assigns local models to local regions of the domain space. A side effect is that the parameter γ can be omitted.

As discussed in [56], another possibility exists to replace the outlier rejection mechanism. Whenever a certain input is not contained by any domain, the model with the smallest e.g. Euclidean distance to the putative outlier could compute the output. In the case of the HED model, the distances $\|x^{(D)} - p\|$ between the input and the positions of the hyper-ellipsoid should be compared. An equivalent definition for the SVD model would compute the distance to the mean of all support vectors of a domain: $\|x - \frac{1}{S}\sum_{i=1}^{S} s_i\|$.

3.4. Local Models

The local models of an HLAM realize the mapping from the model space into the output space. Together they approximate the target function. Each local model is specialized to its domain i.e. it is trained with the same data as its domain. One can decide which supervised machine learning technique is employed to establish the local models. During the training of an HLAM (cf. Section 3.7) the MLT will be executed with training data and is supposed to estimate the parameters of the local model. In the following these parameters are denoted as Θ_{MLT}. A learning algorithm is denoted as `TrainMLT(·)`. As input it receives a set of training samples $T = \{(x_i^{(M)}, y_i)\}$.

In this section the parameters of two different standard machine learning techniques are defined. These two were used in the experiments of Chapter 4. Their learning algorithms are not described in much detail as it would be beyond the scope of this thesis.

Linear and Polynomial Models

The simplest way to approximate a target function is to assume a linear relationship between input and output according to:

$$\hat{y}_{\mathrm{LM}}(x^{(M)}) = \beta^T \tilde{x}^{(M)}, \tag{3.16}$$

where $\beta \in \mathbb{R}^{w+1}$ are the parameters of the linear model (LM) and $\tilde{x}^{(M)} = (x^{(M)}, 1)^T$ is the original input extended by a constant component. The next level of flexibility can be achieved with a polynomial model (PM) of order d as:

$$\hat{y}_{\mathrm{PM}}(x^{(M)}) = \beta_d^T \left(x^{(M)}\right)^d + \cdots + \beta_2^T \left(x^{(M)}\right)^2 + \beta_1^T x^{(M)} + \beta_0, \tag{3.17}$$

where $\beta_{d,\ldots,1} \in \mathbb{R}^w$ and $\beta_0 \in \mathbb{R}$ are the model's parameters. Both models can be trained with the least squares algorithm. Their parameters are given with:

$$\Theta_{\mathrm{LM}} = \beta \quad \text{and} \quad \Theta_{\mathrm{PL}} = (\beta_d, \ldots, \beta_0). \tag{3.18}$$

3.5. Model Validation Criteria

For the training of an HLAM the offline algorithm proposed in Section 3.7 requires a criterion to validate an established local model. Such a criterion should state whether the target function is sufficiently good approximated by a local model in its domain or not. For a general framework this can be formalized as:

$$\text{Eval}(\hat{y}(\cdot), T) = \begin{cases} 1 & : \quad \text{Err}(\hat{y}(\cdot), T) \leq \epsilon \\ 0 & : \quad \text{else} \end{cases}, \tag{3.19}$$

where $\text{Err}(\cdot,\cdot)$ is an error function for the local model $\hat{y}(\cdot)$ on the data set T and ϵ is a manually chosen threshold. The latter serves as an upper bound for an acceptable error in a specific application setting.

The crucial decision remains to choose an error function $\text{Err}(\cdot,\cdot)$. Such a function should measure the approximation performance of the local model. It is typically done by defining a loss function $L(\cdot,\cdot)$ that measures the misfit between a target value y and an estimated output value $\hat{y}(x)$. Common choices for L are:

$$L(y, \hat{y}(x^{(M)})) = \quad (y - \hat{y}(x^{(M)}))^2 \qquad \text{(squared error)}, \tag{3.20}$$

$$L(y, \hat{y}(x^{(M)})) = \quad \|y - \hat{y}(x^{(M)})\| \qquad \text{(absolute error)}, \tag{3.21}$$

$$L(y, \hat{y}(x^{(M)})) = \quad I(y \neq \hat{y}(x^{(M)})) \qquad (0-1 \text{ loss}). \tag{3.22}$$

The squared and absolute error are used for regression problems, while the 1-0 loss is defined for classifications.

The most common error function definition is the *test* or *generalization error* Err_{Test} which is the mean loss

$$\text{Err}_{\text{Test}}(\hat{y}(\cdot), T_{\text{Test}}) = \frac{1}{l} \sum_{i=1}^{l} L(y_i, \hat{y}(x_i x^{(M)})) \tag{3.23}$$

over a test set $T_{\text{Test}} = \{(x_i^{(M)}, y_i)\}$ with $i = 1, \ldots, l$. To do so, the set of all available samples of the target function is split into a training and a test set. As the names indicate, with the first set the parameters of a local model are estimated, while the second set is used to compute the model's test error according to (3.23). The generalization error offers the great benefit that the prediction capability of a local model is estimated on a data set that is independent from the training set. It indicates the approximation performance of a local model by its success to generalize to unseen data.

The problem with Err_{Test} is that it only works well in data rich situations. Only with enough samples the parameters of a local model as well as the generalization error can properly be estimated. This is typically not the case. The HLAM offline learning algorithm complicates this fact as the whole data set is split up for the different domains. Hence the number of samples per domain may become only a small fraction of the original data set. Hence, the strict separation of the training and test sets has to be reconsidered. Another aggravating factor is that the validation process should not be computational

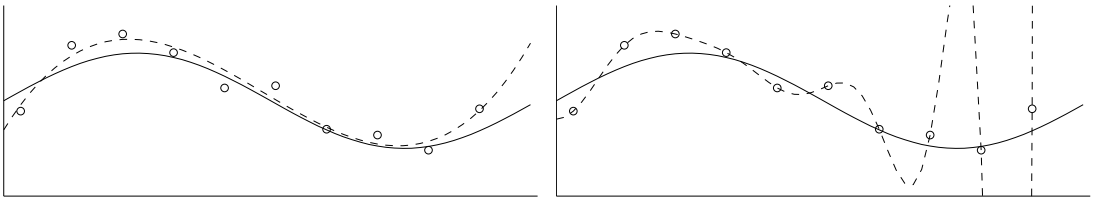


Figure 3.8.: Two approximations (dashed plots) of the same target function (solid plot). The training samples are shown as circles. In panel (a) a polynomial of order 3, in panel (b) one of order 13 is shown.

too demanding since typically many local models have to be checked during the training of an HLAM.

3.5.1. Validation with the Training Set

The easiest but also most venturous possibility to validate a local model's performance is to examine its training error $\mathrm{Err}_{\mathrm{Train}}$. Analog to (3.23) this error is the mean loss over the training set T. Its advantage is that all the available data samples can be employed to estimate the parameters of the local model. The major problem is that the training error is an underestimation of the actually desired test error. Using the same data set for training and testing will privilege over-fitted solutions that generalize badly. Over-fitting becomes especially a problem when the chosen type of local model is very flexible.

This risk with the training error criterion is visualized in Figure 3.8. It shows how a periodic target function is approximated by two polynomials of different orders. Regarding the plotted training samples one can see that the polynomial of smaller order yields a higher training error than the other polynomial which realizes a perfect fit. But obviously the less flexible polynomial generalizes better over the training set. So in the example, a validation criterion should accept the polynomial of smaller order and reject the other one. With the training error criterion this can not be ensured. Still, in the case where the local model matches the structure of target function, this criterion will be as good as the test error criterion.

3.5.2. Validation by Cross-Validation

In the absence of enough data sample cross validation is a very well known technique to estimate the generalization error. For a K-fold cross validation the data set is split into K roughly equal-sized subsets. The idea is that each of the K subsets should serve once as a test set. K local models are trained so that the K^{th} model is fitted to all the data samples except those from the K^{th} subset. Consecutively, the test error $\mathrm{Err}_{\mathrm{Test}}$ of the K^{th} model is computed on its corresponding subset. The result of the cross validation is

then given as:

$$\mathrm{Err}_{\mathrm{CV}}(T) = \frac{1}{K}\sum_{i=1}^{K}\mathrm{Err}_{\mathrm{Test}}(\hat{\mathrm{y}}_{\mathrm{i}}(\cdot),T_i), \tag{3.24}$$

where T_i are the K generated subsets and the local models $\hat{y}_i(\cdot)$ are trained on $T \setminus T_i$.

A value for K has to be chosen. In general one can say that the higher the value the more trustworthy the result of the validation will be. The extreme case where K equals the number of data samples is known as *leave-one-out* cross validation. Beside this, in the literature most often a five or ten-fold cross validation is found. For more insights about cross validation see [52].

The biggest advantage of the cross validation criterion is that the training sets are strictly separated from the test sets. That reduces the danger of accepting over-fitted solutions. Those local models that may perfectly fit the training data but that generalize badly over the test sets will be rejected. Hence, cross validation is especially well suited if a very flexible type of local model was chosen. On the other hand, the computational burden can easily become substantial. If the value of K is high and the learning algorithm for a local model is complex, cross validation may cause practical problems during the training of an HLAM.

3.6. Algorithm to Unify Domains

As outlined on page 24 the approximation performance of an HLAM can be increased by the unification of local models of the network. In Subsection 3.3.4 two methods were given how this can formally be realized. In the following the question will be tackled which domains of an HLAM should be combined. First, the so-called *neighborhood graph* will be proposed to express which domains are adjacent to each other. Upon this, an algorithm is presented that actually unifies the domains. Finally, different criteria can be defined that select domains to be combined.

3.6.1. Neighborhoods of Domains

A unification process in the HLAM approach must adhere to its basic principle that domains of local models represent local regions of the input space. So, only adjacent domains may be unified. In this thesis adjacency is expressed with the neighborhood graph that is similar to the graph generated in the DCS approach (cf. Section 2.4). Both graphs are based on the same idea that two domains are regarded as adjacent to each other if data samples are located next to them. The neighborhood graph G can be defined for a set of M domains and a set $T = \{x_i^{(D)}\}$ of input samples. The graph G is given as a symmetric $\mathbb{N}^{M\times M}$ matrix in which the components $G_{k,l}$ and $G_{l,k}$ equal the number of input samples $x_i^{(D)}$ that are located next to the k^{th} and the l^{th} domain. To decide whether a sample is located next to a domain, the Euclidean distance[2] between the input sample $x^{(D)}$ and a reference point $r(\Phi, x^{(D)})$ of the domain is computed and compared

[2] In fact, a user can choose which distance measure is the most appropriate one for a specific application.

Algorithm 3.4: Pseudo code for computing the neighborhood graph given a training set and a set of domains.

Function `EstablishNeighborhoodGraph`
Input : $T = \{(x_i^{(D)}, y_i)\}, \{\Phi_l\}$,
Output: $G \in \mathbb{N}^{M \times M}$
begin
  $G \leftarrow 0$
  forall i **do**
    $k_1 \leftarrow \arg_l \min \|x_i^{(D)} - r(\Phi_l, x^{(D)})\|$
    $k_2 \leftarrow \arg_{\{l|l \neq k_1\}} \min \|x_i^{(D)} - r(\Phi_l, x^{(D)})\|$
    $G_{k_1,k_2} \leftarrow G_{k_1,k_2} + 1$
    $G_{k_2,k_1} \leftarrow G_{k_2,k_1} + 1$
end

with the distances to other domains. If the computed value is the smallest or the second smallest distance, the sample is regarded as located next to the corresponding domains. In other words, a data sample is located next to those domains that are the nearest and second nearest neighbors in the domain space. Depending on the domain model different definitions of a reference point to a domain are used according to:

$$r(\Phi, x^{(D)}) = \begin{cases} p : \Phi_{\text{CD}} = p \\ p : \Phi_{\text{HED}} = (p, s_1, \ldots, s_v, w_1, \ldots, w_v, \gamma) \\ \frac{1}{S} \sum_{i=1}^{S} s_i : \Phi_{\text{SVD}} = (s_1, \ldots, s_S, \alpha_1, \ldots, \alpha_S, \gamma) \\ \arg_{r(\Phi_i, x^{(D)})} \min \|x^{(D)} - r(\Phi_i, x^{(D)})\| : \Phi_{\text{UNI}} = (\Phi_1, \ldots, \Phi_r) \end{cases}, \quad (3.25)$$

For the CD and HED model the reference point is the prototype and position parameter, respectively. Whereas the mean of all support vector defines the reference point to a support vector domain. The distance to a domain that is unified by association is computed: the reference point of the associated domain that is nearest to the sample is taken as the reference point of the unified domain. Algorithm 3.4 gives the pseudo code for establishing a neighborhood graph.

The matrix G expresses two things: if a component is non-zero, the domains corresponding to its indices are neighbors in the domain space. In Figure 3.9, this neighborhood relation is symbolized as an edge between those domains. Secondly, the value of the components defines the strength of such an edge. It indicates how much evidence were found in the data set for supporting the neighborhood relation. This knowledge is an extension to the neighborhood graph definition of the DCS approach and is especially useful to guide the unification process as described below. The main difference to the SOM neighborhood concept (cf. Section 2.3) remains that the adjacency of domains is established purely data driven. The topology of domains has not to be pre-specified. As visible in Figure 3.9 the definition of G allows a variety of neighborhood structures: both, very densely connected domains and isolated structures with at least two domains are possible.

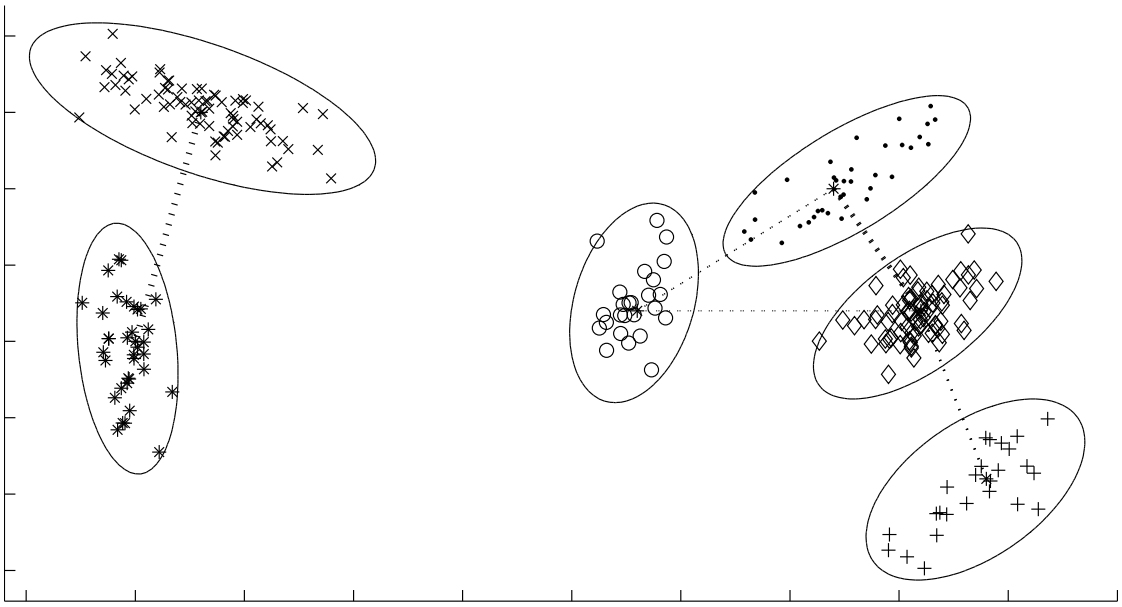


Figure 3.9.: Example of a neighborhood structure of six hyper-elliptical domains. Domains that are adjacent to each other according to the neighborhood graph are connected with a dashed line. The thickness of the lines indicates how well the data supports this neighborhood structure.

3.6.2. Recursive Algorithm to Unify Domains

With a neighborhood graph computed for a given HLAM one can easily identify those domains that can be unified without violating the principle of locality in the HLAM approach: all those two domains are proper candidates for a unification that correspond to a non-zero component of the graph matrix G. The algorithm for unifying domains works on this set of candidates. Its goal is to find and unify a pair of candidate domains so that the performance of the whole HLAM is improved.

To do so, for each pair of candidates the available training samples of two candidate domains are unified and used as the data set for the training and validation of a new local model. The training and validation is performed with methods discussed in Section 3.4 and Section 3.5, respectively. The validation of each new model yields an error term that is compared with the error terms of the other local models. Only this model gets finally accepted that produces the smallest error and passes the chosen model validation criterion. To finalize such a unification, the two former local models are replaced in the HLAM by the new one. To do so, the two candidate domains are unified by one of the methods proposed in Subsection 3.3.4 (e.g. by association). The result will be an HLAM that contains one local model less than before and still meets the chosen quality criterion. Upon this the whole unification process can be repeated by an recursive invocation. The process will stop when no more domains can be unified without violating the model validation criterion. Its pseudo code is given in Algorithm 3.3.5.

Algorithm 3.5: Pseudo code for the algorithm that unifies domains.

Function `RemoveRedundantModels`
Input : $M^{in} = \{m_k = (\Theta_k, \Phi_k, T_k)\}$
Output: $M^{out} = \{(\Theta_k, \Phi_k, T_k)\}$
begin
  $G \leftarrow$`EstablishNeighborhoodGraph` $(\bigcup_k T_k, \bigcup_k \Theta_k)$
  $\{(i,j)\} \leftarrow \mathrm{SC}(G)$
  foreach (i,j) **do**
    $T_{i,j} \leftarrow T_i \cup T_j$
    $\Theta_{i,j} \leftarrow$`TrainMLT` $(T_{i,j})$
    $\epsilon_{i,j} \leftarrow \mathrm{Err}(\Theta_{i,j}, T_{i,j})$
  $(k,l) \leftarrow \arg_{i,j} \min \epsilon_{i,j}$
  if $\mathrm{Eval}(\Theta_{k,l}, T_{k,l})$ **then**
    $\Phi_{k,l} \leftarrow$`UnifyDomain`$(\Phi_k \cup \Phi_l)$
    $M \leftarrow (M^{in} \setminus \{m_k, m_l\}) \cup \{(\Theta_{k,l}, \Phi_{k,l}, T_{k,l})\}$
    $M^{out} \leftarrow$`RemoveRedundantModels` (M)
  else
    $M^{out} \leftarrow M^{in}$
end

In its simplest form (as described above) this algorithm may become computational very demanding. If the HLAM contains many local models, its neighborhood graph will contain many non-zero components so that many pairs of candidate domains have to checked for one successful unification. In a case where the training of a local model is very expensive due to the chosen machine learning technique the unification process will be infeasible. A better strategy will select from the set of all candidates these domains that are according to some heuristic the most promising ones for a unification. With such a selection mechanism (denoted as "SC(·)" in the pseudo code) it is possible to introduce rules that promote certain neighborhood structures. In the next section different selection criteria will be discussed.

After clarifying how the proposed unification algorithm works, one can argue when it should be used. Formally, it can be invoked for any HLAM that contains more than at least two domains. One can decide depending on an HLAM at hand when it should be tested if domains can be unified. In cases where a trained HLAM has to be analyzed it is especially advised to try to simplify the realized solution by unification. The decrease of local models can also improve the generalization performances as the danger of overfitting with too many models is reduced. Hence in an offline learning scenario, the unification algorithm should be performed one time directly after training, while during online learning it could be started in regular intervals. Experiments described in Section 4.7 are exemplifying this.

3.6.3. Selection Criteria to Unify Domains

To speed up the unification process, the set of candidate domains should be reduced from all possibilities the neighborhood graph allows. In the most extreme case where a graph with M domains is fully connected (i.e. where every domain is adjacent to each other) $\binom{M}{2}$ local models have to be trained and validated. The worst case can only occur when the number of local models is quite small in relation to the number of dimensions of the domain space (e.g. in 1D domain space a graph with only 2 domains can be fully connected). Still the possibilities that two domains are adjacent to each other grow rapidly with the number of local models in an HLAM and the dimensionality of the domain space. Hence for practical reasons it is very advisable to define criteria to select only a small subset of candidates the neighborhood graph offers.

Every selection criterion is defined as a function of the neighborhood graph that outputs a set of unordered pairs of indices to the domains that should be unified. The pairs are unordered i.e. $(i, j) = (j, i)$ since it does not make any difference if the i^{th} domain is unified with the j^{th} or visa versa. The most unrestricted (but as discussed the most expensive selection) is to take all candidates that are valid according to the neighborhood graph:

$$\text{SC}_{\text{All}}(G) = \{(i, j) \mid G_{i,j} \neq 0\}. \tag{3.26}$$

Since every possible unification of two domains will be tested this selection criterion has the advantage that the local model that is optimal w.r.t. the chosen validation criterion is added to the HLAM.

One goal of the unification process is to increase the robustness of the local models. This should be achieved whenever two domains are unified as this increases the training set of a local model. This principle grounds the idea that those domains should be promoted as candidates that increase the training set at most. This can be formalized with:

$$\text{SC}_{\text{Max}}(G) = \{(i, j) \mid i = 1, \ldots, M \ \wedge \ j = \arg_k \max G_{i,k}\}. \tag{3.27}$$

It selects for each domain this domain for a unification that has the maximal number of data samples as their common nearest neighbors. For an HLAM with M domains this criterion picks at most $M - 1$ candidate pairs. The number of candidates is smaller in cases where two domains have each other as the domains with the most common data samples. The two domains in the upper left corner of Figure 3.9 represent such a case.

The other goal of the unification process is to reduce the number of local models as much as possible. A unification may often fail due to the inflexibility of the used type of local model to cope with the enlarged training set. With e.g. a simple linear model it may not be possible to approximate the target function with the demanded quality if the unified domain contains too many too different training samples. A strategy to avoid this problem is to try to unify only those domains with a small number of training samples. This can be defined with

$$\text{SC}_{\text{Min}}(G) = \{(s, j) \mid \forall j : G_{s,j} \neq 0\}, \tag{3.28}$$

where s is the index of the domain that contains the minimal number of training samples (i.e. $s = \arg_i \min |T_i|$). A more restrictive specification selects only this single neighbor of the domain s that has the maximal number of training samples as nearest neighbors in common:

$$\mathrm{SC}_{\mathrm{MinMax}}(G) = \{(s, \arg_j \max G_{s,j})\}. \tag{3.29}$$

Another unification strategy is to concentrate effort on this domain that is strongly embedded in a neighborhood structure. The conception is that a domain that has many neighbors may be the result of a splitting process that divided the domain space into too small local regions. Recombination of such a domain with one of its neighbors might improve the approximation performance.

The notion of being strongly embedded in a neighborhood structure can be defined in two plausible ways. Either a domain has many neighbors or many data samples are located next to it. With $a(\cdot)$ denoting the index of the sought domain the first idea can be formalized as:

$$a(G) = \arg_i \max |\{j \mid G_{i,j} \neq 0\}| \tag{3.30}$$

where $\{j|G_{i,j} \neq 0\}$ is the set of column indices of non-zero components of the i^{th} row of the neighborhood matrix G, i.e. it is the set of indices of the neighbors of the i^{th} domain. The alternative is defined with:

$$a(G) = \arg_i \max \sum_{j=1}^{M} G_{i,j}, \tag{3.31}$$

where this row index i is chosen which maximizes the sum of all the components of the i^{th} row.

With $a(\cdot)$ the selection criterion can be given as:

$$\mathrm{SC}_{\mathrm{Cluster}}(G) = \{(a(G), j) \mid \forall j : G_{a(G),j} \neq 0\}, \tag{3.32}$$

which is the set of pairs of the domain with index $a(G)$ and all its neighbors. This heuristic rule is denoted as $\mathrm{SC}_{\mathrm{Cluster}}$ as it tends to produce clusters of domains. Since the unified domain remains adjacent to its former neighbors it will be selected again as a candidate in the recursive call of the unification algorithm. Hence the domain that was chosen in the first step of the unification process will be extended with its adjacent domains as long as the training of the new local model is successful.

3.7. Learning Algorithms to Built up an HLAM

In the last five sections different techniques were proposed that solve different subproblems during the training process of an HLAM. These techniques constitute partial solutions of the complete training. They serve as modules in the learning algorithm presented in this section. Since alternative solution for the different subproblems were given one

can decide what characteristics an HLAM should have to cope successfully with a certain learning problem. In the following an offline and an online learning algorithm is presented that build on the these techniques.

3.7.1. Offline Learning

As already outlined in Subsection 3.1.2.2 the basic idea for offline learning is to divide the training set into subsets of samples that belong to a local region and can be approximated by one local model. The needed assignment of samples to a local region resembles a clustering process. But different to normal clustering approaches that only regard the input values of the samples the new learning algorithm must also take the output values into account. Otherwise the actual task of approximating the target function can not be accomplished. Furthermore, a trained HLAM should contain as less local models as possible. This facilitates an interpretation of a network.

The new learning algorithm is a recursive process that divides the training set by means of a clustering mechanism into subsets. With each recursive invocation of the process a subset is divided into further subsets on a finer scale. For each established subset a local model is trained and validated. If the target function is sufficiently good approximated, the local model is added to the HLAM network together with its domain. The domain is established with one of the methods described in Section 3.3. If the approximation performance is not satisfying and the subset contains enough training samples, this process of dividing and training is started again with a recursion.

In more detail: the learning algorithm (given as pseudo code in Algorithm 3.3.6) starts with one local model having the whole input space as its domain. Using all available training samples the model's parameters Θ are estimated with the chosen machine learning technique `TrainMLT`$(\cdot)$. Then the new model is validated with one of the proposed error criteria. If the model approximates the target function good enough, it will be added to the network. Its domain will be established with a method presented in Section 3.3. Such a technique will estimate the domain's parameters Φ given the same data samples as the local model. Otherwise, if the model's approximation performance is not acceptable and the number of its training samples exceeds a manually chosen threshold $minSamples$, the model will be rejected and its set of training samples is split into new subsets. This splitting is realized by a special clustering method described below. For each gained subset the same process of training a model, its validation and the possible splitting is recursively invoked. This process will be repeated until all local models perform sufficiently good or a set of training samples can not be split any more.

A distinctive feature of the algorithm is how a set of training samples T will be split up when the approximation performance of a trained model $\hat{y}_\Theta(\cdot)$ is not satisfying. A method was implemented which clusters the input samples w.r.t. the error $\epsilon_j = \|\hat{y}_\Theta(x_j) - y_j\|$ of the local model for the sample x_j. This is done with an iterative scheme that determines a number of L cluster centers μ_l by minimizing:

$$\sum_{l=1}^{L} \sum_{x \in S_l} \|x - \mu_l\|^2 \text{ with} \tag{3.33}$$

$$S_l = \{x_j \in T \mid l = \arg_i \min \|x_j - \mu_i\|^2\} \text{ and} \tag{3.34}$$

$$\mu_l = \sum_{x_i \in S_l} \frac{\epsilon_i}{\sum_j \epsilon_j} x_i \tag{3.35}$$

W.r.t. these cluster centers μ_l the training samples are distributed to the subsets T_l according to a nearest neighbor decision:

$$T_l = \{(x_j, y_j) \in T \mid l = \arg_i \min \|x_j - \mu_i\|^2\}.$$

This minimization process is essentially the well known k-means algorithm [8, 78]. In its common implementation the cluster centers μ_l are set to the arithmetic mean $\frac{1}{|S|}\sum_{x \in S} x$. In contrast to that, the proposed algorithm uses the errors ϵ_j to calculate the cluster centers according to (3.35). The normal k-means algorithm would generate clusters only driven by the distribution of the samples in the input space. Whereas the weighted version ensures that the generated subsets are centered in regions where the performance of model $\hat{y}_\Theta(\cdot)$ was bad. This resembles the learning strategy of the well known *AdaBoost* algorithm [42]: training is emphasized on samples that yield repeatedly high approximation errors. AdaBoost is an iterative scheme that generates resamplings of the target function and trains and validates models for these. Such resamplings mainly contain those samples that were not successfully learned in former iterations. Similarly in the HLAM offline learning algorithm, samples that cause high errors have strong impact on the results of the clustering process. Hence more local models will be concentrated in regions of the input space that contain samples that are apparently hard to learn.

The proposed recursive divide and conquer strategy realizes the needed coupling between the clustering and approximation task. The clustering process is driven in two ways by the success or failure of the approximation of the target function. On one hand side, a local region is divided into more domains when one local model is not able to realize a satisfying approximation quality. On the other hand, the weighted clustering process itself concentrates domains where it is hard to approximate the target function. Furthermore, the integrated model validation step ensures that new local models are only added to an HLAM when it is necessary. This promises that the overall number of local models is kept small and hence the interpretation of a trained network is feasible. On the other side, the repeated training process can result in practical problems if the chosen machine learning technique is computational expensive (e.g. because it depends heavily on fine tuning of its meta-parameters).

3.7.2. Online Learning

The online learning algorithm is designed to be used in applications where training data is only available as a continuous stream of samples. The idea is that the input space of the target function is being explored during the run-time of the system. In the ideal case, the stream of samples forms a continuous trajectory through the input space. But the online learning algorithm should also be able to cope with an arbitrary sampling scheme of the target function. Furthermore an HLAM should capture dynamic changes

Algorithm 3.6: Pseudo code for the offline learning algorithm.

Function `TrainHLAM`
Input : $T = \{(x_j, y_j)\}$
Output: $M = \{(\Theta_k, \Phi_k)\}$
begin
  $m \leftarrow$`TrainModelAndDomain(`T`)`
  $M \leftarrow$`OptimizeModel(`m, T`)`
end

Function `TrainModelAndDomain`
Input : $T = \{(x_j, y_j)\}$
Output: $m = (\Theta, \Phi)$
begin
  $\Theta \leftarrow$ `TrainMLT(`T`)`
  $\Phi \leftarrow$ `EstablishDomain(`T`)`
end

Function `OptimizeModel`
Input : $m = (\Theta, \Phi), T = \{(x_j, y_j)\}$
Output: $M = \{(\Theta_k, \Phi_k)\}$
begin
  if $\neg\text{Eval}(\hat{y}_\Theta(\cdot), T) \wedge |T| > L{\cdot}minSamples$ **then**
    $\{T_l\} \leftarrow$`SplitTrainingSet(`T, Θ`)`
    forall l **do**
      $m_l \leftarrow$`TrainModelAndDomain(`T_l`)`
      $M_l \leftarrow$`OptimizeModel(`m_l, T_l`)`
    $M \leftarrow \bigcup_l M_l$
  else
    $M \leftarrow \{m\}$
end

Function `SplitTrainingSet`
Input : $T = \{(x_j, y_j)\}, \Theta$
Output: $R = \{T_l\}_{l=1,\ldots,L}$
begin
  forall j **do**
    $Err_j \leftarrow \|\hat{y}_\Theta(x_j) - y_j\|$
  $\{\mu_l\}_{l=1,\ldots,L} \leftarrow \min \sum_{l=1}^{L} \sum_{x_j \in S_l} \|x_j - \mu_l\|^2$ with

$$\mu_l = \sum_{x_i \in S_l} \frac{Err_i}{\sum_j Err_j} x_i \quad \text{and}$$

$$S_l = \{x_j \in T \mid l = \arg_i \min \|x_j - \mu_i\|^2\}$$

  forall l **do**
    $T_l \leftarrow \{(x_j, y_j) \in T \mid l = \arg_i \min \|x_j - \mu_i\|^2\}$
end

of the target function. This is different to the offline learning scenario where a certain input value always corresponds to a certain output value. The online learning algorithm should adapted an HLAM to a function that may change over time.

Two basic ideas ground the development of the online learning algorithm. First, the activation function of a domain defines the responsibility of the corresponding local model to a given training sample. This information is used to decide which local model should be adapted to the new training sample. Second, the neighborhood graph is used to divide the domain space into local regions w.r.t. established domains. This is important to decide where new domains should be established. The pseudo code is given in Algorithm 3.3.7. It defines a procedure `UpdateHLAM` that should be called whenever a new training sample (x, y) is available. The algorithm works only with the HED or SVD model. The reason will be given further below.

The algorithm works as follows: an HLAM is initialized with no local model at all. Like the offline version, the online algorithm has a meta-parameter *minSamples* that specifies how many samples at least have to be assigned to one local model. After initialization, this manually chosen number of samples have to be collected before the first local model and its domain will be added to the HLAM. To do so, new samples are temporary stored in a buffer T^t. Throughout the run-time of the system, this buffer will contain all the samples that could not be assigned to a local model of the HLAM. If the size of buffer T^t equals *minSamples*, the first model and its domain will be established with all the sample from the buffer. The model's and the domain's parameter Θ^t and Φ^t, respectively, are estimated by the same means as in Algorithm 3.3.6. Along with Θ^t and Φ^t an extra buffer T^t containing all the used training samples is added to the HLAM. This buffer T^t represents the sample history of each local model. How it works in detail will be explained down below.

After the first model was added to an HLAM, for each new training sample it will be decided if an already existing local model should be updated or a new model should be added to the HLAM. This decision depends on whether the sample could be successfully approximated or not. So, the minimal loss $L(\hat{y}_{\Theta_i^t}(x), y)$ of all local models of an HLAM is determined and compared with a manually chosen threshold ϵ. This is the same idea to test the success of the HLAM training as applied in the offline learning algorithm and based on the definitions given in Section 3.5.

If the approximation performance is good enough, the new sample is used to update one local model. Before the question which model should be updated can be settled one should first discuss how it is updated: As noted above, the online algorithm stores for each local model the history of samples that were assigned to the model during run-time. The samples are collected in a buffer T^t which serves as a single training set. The update process first adds the new sample to T^t and then uses the same means as the offline learning algorithm to train with T^t the local model and its domain.

Obviously, the usability of the algorithm would be quite limited if the size of the buffer can be arbitrarily large. The computational costs in time and memory could easily become exhausting. The buffer could grow arbitrarily as long as the system keeps running and is collecting new samples. As a result of too many training data, the re-training of the local models and domains would become too time demanding. Besides such practical

Algorithm 3.7: Pseudo code for the online learning algorithm.

Function `UpdateHLAM`
Input : $(x, y), M^t = \{(\Theta_k^t, \Phi_k^t, T_k^t)\}, B^t$
Output: $M^{t+1} = \{(\Theta_k^{t+1}, \Phi_k^{t+1}, T_k^{t+1})\}, B^{t+1}$
begin
  if $M = \emptyset$ **then**
    $B^{t+1} \leftarrow B^t \cup \{(x, y)\}$
    if $|B^{t+1}| \geq minSamples$ **then**
      $(\Theta^{t+1}, \Phi^{t+1}) \leftarrow$`TrainModelAndDomain` (B^{t+1})
      $M^{t+1} = (\Theta^{t+1}, \Phi^{t+1}, B^{t+1})$
      $B^{t+1} = \emptyset$
  else
    $Err \leftarrow \min_i L(\hat{y}_{\Theta_i^t}(x), y)$
    if $Err \leq \epsilon$ **then**
      $l \leftarrow \arg_i \max d_{\Phi_i^t}(x^{(D)})$
      $(\Theta_l^{t+1}, \Phi_l^{t+1}, T_l^{t+1}) \leftarrow$`UpdateLocalModel`$(T_l^t, (x, y))$
    else
      $l \leftarrow \arg_i \max a_{\Phi_i^t}(x^{(D)})$
      if `SampleBelongsToDomain`(x, Φ_l^t) **then**
        $(\Theta_l^{t+1}, \Phi_l^{t+1}, T_l^{t+1}) \leftarrow$`UpdateLocalModel`$(T_l^t, (x, y))$
      $B^{t+1} \leftarrow B^t \cup \{(x, y)\}$
      $G \leftarrow$`EstablishNeighborhoodGraph` $(B^{t+1}, \{\Phi_k^t\})$
      $(l, s) \leftarrow \arg_{i,j} \max G_{i,j}$
      if $G_{l,s} \geq minSamples$ **then**
        $T^{t+1} \leftarrow$`GetAllSamplesBetweenDomains`(B^{t+1}, l, s)
        $B^{t+1} \leftarrow$`RemoveAllSamplesFromBetweenDomains`(B^{t+1}, l, s)
        $(\Theta^{t+1}, \Phi^{t+1}) \leftarrow$`TrainModelAndDomain` (T^{t+1})
        $M^{t+1} \leftarrow M^t \cup (\Theta^{t+1}, \Phi^{t+1}, T^{t+1})$
end
Function `UpdateLocalModel`
Input : $T^t, (x, y)$
Output: $(\Theta^{t+1}, \Phi^{t+1}, T^{t+1})$
begin
  $T^{t+1} \leftarrow T^t \cup \{(x, y)\}$
  if $|T^{t+1}| > maxBufferSize$ **then**
    $T^{t+1} \leftarrow$ `RemoveOldestSample`(T^{t+1})
  $(\Theta^{t+1}, \Phi^{t+1}) \leftarrow$`TrainModelAndDomain` (T^{t+1})
end

problems, an HLAM could not adapted to a dynamically changing target function. No sample of an input-output correspondence would be forgotten. Over time, the employed MLT would have to solve the impossible task to cope with data that contradicts itself.

For this reason the size of the buffer T^t is limited by another meta-parameter called *maxBufferSize*. The access to the buffer should be implemented like a queue, so that the oldest sample will be discarded if a new sample is available and the buffer's maximal size is reached. How this can be done exactly is left to a programmer. In the pseudo code, it is encapsulated in the procedure `RemoveOldestSample`. The parameter *maxBufferSize* has to be chosen in accordance with the type of local model, the assumptions about the dynamics of the target function and the available memory and CPU power. The buffer may store rather many samples if the employed MLT demands it due to its complexity. On the other hand, the buffer should be small if the target function is quickly changing and the computational burden has to be kept low. In any case *maxBufferSize* must be larger or equal to the other meta-parameter *minSamples*.

So, the question remains which local model should be updated by this process. The basic idea is to adapt this model to the new sample which domain contains or is the nearest to it. The algorithm selects the model that has the highest value of $d_\Phi(x^{(D)})$, the function that based the definition of the HED and SVD activation function (cf. (3.8) and (3.11)). It is important to use the function $d_\Phi(\cdot)$ instead of the proper activation functions $a_\Phi(\cdot)$ defined with (3.9) and (3.13). These activation functions drop to zero (minus infinity, respectively) at the boundary of a domain. Hence those domains, that do not contain the new sample, have the same activation value. So, it can not be decided which domain is the nearest to the sample. Only domains that enclose the new sample, hence overlap each other, could be discriminated with their activation function value.

But if only such domains would be taken into account for an update, domains could never expand, instead their size would shrink over time. The reason for that is grounded in the update process with the history buffer. The size of a domain is determined by the samples that are located at the boundary of the domain. But these will eventually be pushed out of the history buffer by samples that must lay inside of the domain to be accepted for an update. Hence the new samples from within the former domain will define the new but tightened boundary. In contrast to that, a domain can expand if samples are used for an update that are in the vicinity and not necessarily inside of it. This conception is implemented by the selection mechanism that updates the model that has the highest value of $d_\Phi(x^{(D)})$.

If the approximation performance of an HLAM is not satisfying for the new sample, the online algorithm improves it by following two ideas. If a local model already exists that should handle the new sample, this model will be updated with it. Additionally, the structure of the network may be changed by inserting a new local model. The first idea is implemented in a straightforward manner taking advantage of the sharp boundaries of the HED or SVD model. The model with the highest activation value $a_\Phi(x^{(D)})$ is checked if its domain contains the new sample, and updated in such a case. To ease the understanding of the pseudo code, the procedure `SampleBelongsToDomain` is used although not explicitly defined in the code. The procedure only encapsulates the domain model specific parameters needed to decide if a sample is included by a domain or not.

The second idea needs more effort to be implemented: it relies essentially on the neighborhood graph defined in Subsection 3.6.1. The graph offers the advantage that a set of samples can be grouped w.r.t. already established domains of the HLAM. It clusters samples together that have the same domains as their nearest neighbors, hence are located in the same region of the domain space. Given G, the algorithm inserts new domains in those regions where enough samples could be collected. To do so, every new sample that is not approximated good enough is first stored in the extra buffer B^t. Then the neighborhood graph G is computed for these samples with Algorithm 3.4.[3] If there exists a component $G_{l,s}$ that is equal or larger than the threshold *minSamples*, a new local model is created with a domain that is located in the vicinity of the I^{th} and s^{th} domain. The samples belonging to this local region are removed from the buffer B^t and used as a training set for the new local model and its domain. Again, to simplify the pseudo code the definitions of the procedures `GetAllSamplesBetweenDomains` and `RemoveAllSamplesFromBetweenDomains` are omitted. They only have to manage the samples with an indexing system in correspondence to the domains.

A shortcoming of this buffer is that in theory it may grow unboundedly. Samples will only be removed from the buffer if enough of them are available in the vicinity of two domains. Hence samples may never be used to establish a new local model, hence their information will be lost. The chances that the unused samples are in this sense badly distributed between domains grow with the number of local models. In the worst case the buffer would contain $\frac{1}{2}M(M-1)(minSample-1)$ samples since $\frac{1}{2}M(M-1)$ is the number of possible pairwise combinations of M domains. If this should become a problem in practice, the most straightforward solution would be to define an upper bound for the buffer size and to manage the samples with a queue (like for the buffer of each local model).

Consistent with the basic HLAM idea the online learning algorithm promotes specialization of the local models to local regions of the domain space. Like in the offline version, a sample is assigned to one single model. This implicitly favors the exclusive gating law which assumes that the single models and not a mixture of them are most appropriate to approximate locally the target function. This strict assignment relies on the sharp boundary of the HED and SVD model. That is the reason why the CD model can not be used for online learning.

3.8. Summary

In this chapter the new Hierarchical Network of Locally Arranged Models approach was presented as a coherent framework to create a modular solution to a learning problem. This supervised function approximation technique features two basic properties: The global input space of the target function is divided into local regions where non-redundant models perform the demanded mapping into the output space. Secondly, it is possible to built up heterogeneous hierarchies of such local models in a systematic way. They can be

[3] Remember that the components of G represent the number of samples located in the neighborhood of these domains that correspond to the indices of the component.

Table 3.1.: Overview of the possible options available for the offline and online learning algorithm in the HLAM approach.

Offline Learning Algorithm					
Gating Law	Exclusive			Mixing	
Domain Model	CD		HED		SVD
Model Validation Criterion	$\mathrm{Err}_{\mathrm{Train}}$			$\mathrm{Err}_{\mathrm{CV}}$	
Selection Criterion for Unification	$\mathrm{SC}_{\mathrm{All}}$	$\mathrm{SC}_{\mathrm{Max}}$	$\mathrm{SC}_{\mathrm{Min}}$	$\mathrm{SC}_{\mathrm{MinMax}}$	$\mathrm{SC}_{\mathrm{Cluster}}$

Online Learning Algorithm					
Gating Law	Exclusive			Mixing	
Domain Model	HED			SVD	
Selection Criterion for Unification	$\mathrm{SC}_{\mathrm{All}}$	$\mathrm{SC}_{\mathrm{Max}}$	$\mathrm{SC}_{\mathrm{Min}}$	$\mathrm{SC}_{\mathrm{MinMax}}$	$\mathrm{SC}_{\mathrm{Cluster}}$

heterogeneous because the different subnetworks can work with different input spaces.

In short: The HLAM approach follows a hierarchical divide and conquer strategy in order to approximate complex target functions with sets of comparably simple models. The break down of a complicated problem into simpler subproblems should be beneficial for the attainable performance and the potential analysis of the realized solution.

These basic properties rely essentially on two new developments: the definitions to model the local region (the so-called domain of a single model) and the algorithms to split up the input space appropriately to achieve the needed performance. This is realized with a set of techniques that capture different aspects of the HLAM approach. In this chapter alternative solutions to the different subproblems were presented. This gives a construction kit ready to hand to customize an HLAM for a specific application. One has the choice about the used model definitions and the employed learning algorithms. Regarding the first aspect, one has to decide about a gating law, the domain model and the type of machine learning techniques used for the local models. While the questions dealing with algorithms are concerned with the type of learning scenario (offline or online), the model validation method and the way how domains can be unified in order to improve the performance. An overview of the possible options to configure an HLAM is summarized in Table 3.1.

Two different gating laws were proposed that define the degree of cooperation between different local models to calculate an HLAM's output. The exclusive gating law selects exactly one model per domain to approximate the target function. That simplifies the interpretation of the HLAM but leads to discontinuities at the boundaries of different domains. Exactly the opposite properties has the mixing gating law that computes the amount of contribution of a single model to the overall output relative to all the others.

As domain models three types of different flexibility and computational requirements

were proposed. For each domain model a so-called activation function has to be defined that expresses how valid a local model is for a certain data sample. Additionally, methods were described that – given a set of training samples – estimate the specific parameters of each domain model.

The center domain model is the easiest but also the most limited way to define a domain. It is given as a prototype vector that induces a Voronoi tessellation of the input space as the division into local regions. The center domain model suffers from the linear boundaries between domains that are depending on the positions of adjacent domains. This is different to the hyper-elliptical domain model. It allows an interpretation of a single domain because its parameters define a hyper-ellipsoid with a fixed position and direction and spread of its main axes. The maximal level of flexibility and computational requirements is reached with the support vector domain. It employs a one-class support vector machine that allows highly non-linear domain boundaries. The advantage is that the domain can perfectly be fitted to the training data but that has to be traded for a costly meta-parameter tuning.

As an extension to these three domain models, methods are discussed how single domains can be unified. The goal for that is to reduce the number of needed local models and the danger of overfitting. Since the same number of training samples is distributed over a smaller number of local models these should be estimated more robustly and hence should generalize better to new data. Furthermore the proposed outlier rejection mechanism can secure an application employing an HLAM. With the fixed, sharp boundaries of the hyper-elliptical and support vector domain model it is easily possible to decide whether a given data sample does not match to the training data. In such a case the sample can explicitly be rejected since it stands to reason that the trained HLAM can not cope with it appropriately.

As the last open question concerning the model definition, it is described how different types of machine learning techniques can be used as local model in an HLAM. These are standard techniques as the HLAM approach is not designed for any special type of MLT. Even an HLAM itself could be used a local model. Only practical questions concerning computational requirements or meta-parameter optimization should guide the decision.

Two algorithms were proposed that realize the division of the input space. One is applicable in an offline, one in an online learning scenario. These define the structure of an HLAM i.e. the number of local models and where their domains are located. Both algorithms have to solve a kind of clustering problem that is constrained by the actual goal to approximate the target function. The main strategy for deciding whether and where a new domain should be inserted is driven by the achieved approximation quality. To achieve this, a recursive clustering scheme was developed for the offline learning algorithm that takes the approximation quality of single training samples into account. The desired result is that more local models are inserted in those regions where the target function is more complicated to be approximated. In order to achieve the same effect, the online learning algorithm relies mainly on two ideas: With the activation functions of the domain models it is decided which local model should be adapted to a new sample. On the other hand, the so-called neighborhood graph is used to locate a region of the input space where a new domain will be added. Again, such a structural change of an HLAM is

driven by the achieved approximation quality. Both algorithms feature the advantage to be controlled by a comparably small number of meta-parameters. The offline and online learning algorithms need only two and three meta-parameters, respectively.

Since the learning algorithms define the structure of an HLAM depending on the approximation quality, a method for validating a local model is required. Two criteria were proposed in this chapter. The mean training error is theoretically not as meaningful as the cross validation error. Still, the later is computational very demanding and has to prove in practice its superiority.

The algorithms for the HLAM approach are completed with a method to decide which domains can be unified. As mentioned above, this reduction of local models is beneficial for the approximation quality. The presented algorithm tries recursively to combine adjacent domains according to a set of proposed criteria. The important concept of adjacency of domains is derived from the neighborhood graph. It offers the advantage that the topology of existing domains is established purely data driven. No fixed neighborhood structure has to be manually pre-specified.

Chapter 4

Validation of the HLAM Approach

The HLAM approach as proposed in the last chapter is thoroughly validated in series of experiments with different benchmark tests. The first goal is to ensure that the new method is competitive with known techniques. The second goal is to find practical rules to configure a HLAM for an application. To achieve this, the effects of the different options of the HLAM approach are tested in several experiments.

In Section 4.1 the experimental setup for the three different benchmark test are defined. What follows is the comparison of the HLAM approach with other machine learning techniques. In Section 4.3 tests are described that reveal the advantages and disadvantages of the three different domain models. Thereupon the two proposed gating laws are examined. The effects of different flexible local models and the two model validation criteria are determined in Section 4.5 and 4.6, respectively. Finally, the validation of the domain unification algorithm and the proposed candidate selection criteria are described.

4.1. Benchmark Tests

Three different benchmark tests with their data sets are described in the following. One data set is defined with a function, while the others stem from the free UCI Machine Learning Repository [33]. To ensure a fair comparison of different MLTs, the experimental setups are taken from [68, 74]. These define in detail how the different sets are used for the benchmark tests. In every case, the success of a machine learning technique $\hat{y}(\cdot)$ is measured with the root mean squared error (RMSE) given as

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|\hat{y}(x_i) - y_i\|}, \tag{4.1}$$

for a set of N samples $\{(x_i, y_i)\}$.

4.1.1. Mackey-Glass Data Set

In [68] a benchmark test with the chaotic Mackey-Glass differential equation is described. The experiment stems originally from [87] and defines the task to predict a value of the

time series

$$x(t-1) = (1-a)x(t) + \frac{bx(t-\tau)}{1+x^{10}(t-\tau)}, \tag{4.2}$$

with the parameters chosen to be: $a = 0.1$, $b = 0.2$, $\tau = 17$ and the initial condition as $x(0) = 1.2$. The function $f : \mathbb{R}^4 \to \mathbb{R}$ that has to be approximated is defined as

$$x(t+6) = f\Big(x(t), x(t-6), x(t-12), x(t-18)\Big). \tag{4.3}$$

For a comparison of different machine learning techniques two separate sets of samples of f are generated. One set with $t = 124, \ldots, 1123$ serves as a training set for the MLT, while the other one with $t = 1124, \ldots, 2213$ is used to calculate the RMSE as the final test error.

4.1.2. Abalone Data Set

The goal of the Abalone benchmark test is to estimate the age of an abalone given seven continuous and one discrete attributes. Instead of cutting the shell, staining it and counting the number of rings through a microscope the age should be predicted with attributes that are easier to obtain. So, the sex, length, diameter, height and the weight of four different parts of an abalone are available as input values in the database [33] of 4177 samples. The target output value is discrete and ranges between 1 and 29 years.

As described in [74] the input and output values are normalized to the range $[0.1, 0.7]$. For the training, 3000 samples are randomly selected from the whole data set. The RMSE is calculated for the remaining 1177 sample.

4.1.3. Auto-Mpg Data Set

In [92] the so called Auto-Mpg problem is stated to predict the city-cycle fuel consumption in miles per gallon in terms of three multivalued discrete and four continuous attributes. The data set available at [33] contains 398 samples of this continuous target function. As multivalued discrete input values the number of cylinder, the model year and the origin are given. The continuous attributes are the displacement, horsepower, weight and acceleration of the cars.

The benchmark test is repeated as described in [74]. The input and output values are normalized to the range $[0, 1]$. Since the number of samples is quite limited a number of 50 trials are performed with different training and test sets. For each trial, 320 samples are randomly chosen for training, while the remaining 78 samples are used to compute the RMSE. The mean value of the trials' RMSE is listed as the final result of this benchmark test.

4.2. Validation of the Offline and Online Learning Algorithm

To demonstrate the utility of the proposed algorithms (see Section 3.7) of the HLAM approach, the three above described benchmark tests taken from [68] and [74] are repeated

Table 4.1.: Results of the Mackey-Glass benchmark test.

Method	Number of Hidden Units	RMSE	
		Train Set	Test Test
HLAM offline	92	0.0008	0.0025
HLAM online	98	0.0057	0.0064
GMN	7	0.0100	0.0091
RBF-AFS	21	0.0158	0.0128
OLS	132	0.0107	0.0163

with networks of the same configuration. They use the exclusive gating law (Subsection 3.2.1) together with the hyper-elliptical domain model (Subsection 3.3.2). Linear models (Section 3.4) are taken as local models. The model validation criterion is based on the training error (Subsection 3.5.1). The algorithm to unify domains is not applied. The domain and model space are set equal to the input space as defined with the benchmark tests. The outlier rejection mechanism is not used, instead the method proposed in Subsection 3.3.6 is applied. In the following this configuration will be referred to as the standard configuration of a HLAM.

The meta-parameters of the two learning algorithms are selected by a process of repeated training and testing. Different values for *minSamples,* the error threshold ϵ and *maxBufferSize* (only for the online algorithm) are chosen from an appropriate interval to train a HLAM.[1] The achieved RMSEs on the test set are compared and the best one is cited as the final result. Throughout this chapter, this is the standard way to choose the meta-parameters.

Tables 4.1, 4.2 and 4.3 show the results of the Mackey-Glass, the Abalone and the Auto-Mpg benchmark tests, respectively. For each test the achieved RMSE on the training and the test set is listed together with the number of hidden units. Depending on the employed method these hidden units can be quite different things. E.g. in the case of the HLAM it is the number of local models. The selected values of the meta-parameters for the HLAMs are summarized in Table 4.5.

With the Mackey-Glass data, the HLAM offline and online learning algorithm is compared with three other machine learning techniques: the *Growing Multi Expert* (GMN) [68], the *Orthogonal Least Squares* (OLS) [26], and the *Radial Basis Functions based on the Adaptive Fuzzy System* (RBF-AFS) [27]. Their results are cited from [68]. The performances of the new algorithms on the other two benchmark tests are opposed to the listings in [74]. There, the original *Resource-Allocation Network* (RAN) of Platt [88] is compared to a number of related RBF network techniques: the RANEKF [66] approach combines the RAN with an extended Kalman filter. The *Minimal Resource-Allocating Network* (MRAN) [115] introduces a special pruning method. Its successor MRAN-OLS [74] enhances this by means of a Orthogonal Least Squares algorithm, while the

[1] Although in the Auto-Mpg benchmark test 50 HLAMs have to be trained for 50 different training and test sets the meta-parameters are only optimized once.

Table 4.2.: Results of the Abalone benchmark test.

Method	Number of Hidden Units	RMSE	
		Train Set	Test Set
HLAM offline	22	0.0454	0.0477
HLAM online	19	0.0467	0.0488
RANEKF	144	0.0655	0.0601
GAP-RBF	18	0.0670	0.0613
MRAN	33	0.0764	0.0669
RAN	143	0.0838	0.0768
MRAN-OLS	19	0.0965	0.0901

Table 4.3.: Results of the Auto-Mpg benchmark test. For the 50 trials the mean and standard deviation of the number of hidden units and the RMSE are given.

Method	Mean Number of Hidden Units	Mean RMSE	
		Train Set	Test Set
HLAM offline	3.00 ± 0.00	0.0735 ± 0.002	0.0804 ± 0.008
HLAM online	8.2 ± 0.70	0.0754 ± 0.004	0.0850 ± 0.010
MRAN	4.46 ± 0.74	0.1086 ± 0.010	0.1376 ± 0.023
RANEKF	5.14 ± 0.90	0.1088 ± 0.012	0.1387 ± 0.029
GAP-RBF	3.12 ± 0.75	0.1144 ± 0.013	0.1404 ± 0.027
MRAN-OLS	2.10 ± 0.30	0.1523 ± 0.010	0.1471 ± 0.011
RAN	4.44 ± 0.84	0.2923 ± 0.081	0.3080 ± 0.092

GAP-RBF network [59] is based on new growing and pruning strategies.

The tables show the superiority of the HLAM approach over the other methods w.r.t. the achieved RMSE in all benchmark tests. Both, the offline and the online learning algorithm performed very well. Especially in the Mackey-Glass benchmark test, the offline HLAM realizes a clearly better result than the alternatives. In every case the proposed offline learning algorithm outperforms the online version. So, the general expectation that an offline learning scenario is easier to handle with a MLT is met. It can be pointed out that the new HLAM online learning algorithm exceeds the approximation quality of such offline algorithms like e.g. RANEKF or MRAN.

The interpretation of the shown outcomes w.r.t. the needed number of hidden units is not as simple as for the RMSE. First of all, the same methods behave differently for the different benchmark tests. For the Mackey-Glass test the HLAM needs much more local models than its next best competitor, i.e. 96 vs. 7. On the other hand, in the Abalone case the HLAM is almost as small as the smallest alternative method. In two of three cases the online HLAM algorithm creates more local model than the offline version. Furthermore, the discussion of the needed number of hidden units is rather delusive as these units are – depending on the MLT – quite different things. E.g. in the case of the

Table 4.4.: Results of different domain models for three benchmark tests.

		Mackey-Glass		Abalone		Auto-Mpg	
		Offline	Online	Offline	Online	Offline	Online
Test RMSE	CD	0.0045	×	0.0463	×	0.0807 ± 0.012	×
	HED	0.0025	0.0064	0.0477	0.0488	0.0804 ± 0.008	0.0850 ± 0.011
	SVD	0.0015	0.0048	0.0471	0.0477	0.0793 ± 0.008	0.0872 ± 0.014
Number of Local Models	CD	99	×	22	×	5.00 ± 0.00	×
	HED	92	98	22	19	3.00 ± 0.00	8.20 ± 0.70
	SVD	100	123	22	20	2.00 ± 0.00	8.88 ± 0.72

GAP-RBF one compares RBF neurons with the linear models of the trained HLAMs. Moreover, the goal of the meta-parameter selection process for the HLAMs is to achieve the best RMSE and not the smallest number of local models.

A valid question about the offline learning algorithm is how robustly it generates a unique HLAM solution for a given training set. Since its implementation utilizes the modified k-means clustering it contains an indeterministic component. The result of the clustering method depends on the initial choice for the cluster centers which are selected randomly from the set of training samples. To test if this dependency has an effect on the generated HLAM, the Abalone benchmark test is repeated 50 times with the offline learning algorithm on the same training set. This benchmark test is chosen as its training set is the largest and hence the clustering should be the most unstable. The mean RMSE is 0.0508 with a standard deviation of 0.0008. The number of local models ranged between 21 and 23, its mean is 22.14. These number indicate that the indeterministic component in the learning algorithm is negligible in practice.

4.3. Comparing the CD, HED and SVD Domain Models

The most important decision for constructing a HLAM is to choose a domain model. It has effects on the approximation performance and the interpretability of a trained HLAM. Furthermore, the computational requirements are different. In Section 3.3 the center, the hyper-elliptical and the support vector domain model are proposed. To give some rules for a decision these models are validated with the benchmark tests. Like in the last section, the standard configuration of a HLAM and meta-parameter selection process is applied. Only the domain model is changed.

Table 4.4 presents the results of 15 different tests. Each domain model is used with the offline learning algorithm for all three benchmark tests. The online version could only be tested with the HED and SVD model (cf. Subsection 3.7.2). For every trained HLAM the RMSE on the test set and the number of created local models is listed. The SVD model is based on the radial basis kernel function defined with (3.12) on page 39. For the offline learning algorithm the kernel width σ and the meta-parameter ν of the one-class

Table 4.5.: Values of meta-parameters of the HLAMs with different domain models used for the benchmark tests.

	Meta-Parameters	Mackey-Glass			Abalone			Auto-Mpg		
		CD	HED	SVD	CD	HED	SVD	CD	HED	SVD
Offline	*minSamples*	2	6	2	100	100	100	50	75	150
	ϵ	.001	.001	.001	.0005	.005	.0001	.0005	.0005	.0005
	σ	×	×	0.01	×	×	0.01	×	×	0.5
	ν	×	×	.0001	×	×	0.01	×	×	0.1
Online	*minSamples*	×	5	5	×	100	100	×	25	25
	maxBufferSize	×	50	20	×	100	150	×	50	50
	ϵ	×	.0005	.0001	×	.0005	.0001	×	.004	.001
	σ	×	×	0.01	×	×	0.01	×	×	0.5
	ν	×	×	.0001	×	×	0.01	×	×	0.1

support vector machine learning algorithm is selected with the same method as the other meta-parameters. The found values are also used in the online learning scenario. All applied meta-parameters are given in Table 4.5.

First, one should note that the results achieved with all three domain models are better than those of other approaches (cf. Table 4.1, 4.2 and 4.3). Second, the different domain models lead to quite comparable solutions both w.r.t. obtained RMSE and number of local models. Most clearly only the SVD and the CD model can be differentiated: for the Mackey-Glass benchmark test the SVD model outperforms the CD model in approximation quality, and for the Auto-Mpg data set in the needed number of local models. Still the center domain model has a small advantage in the Abalone test.

As already seen in the last section for the HED model, in the online learning scenario also the SVD model does not perform as well as in the offline scenario. W.r.t. the RMSE the SVD model shows superiority over the HED model in the Mackey-Glass and the Abalone benchmark test. Taking the standard deviation of the 50 trials into account, the results for the Auto-Mpg are very similar. In case of the Mackey-Glass data set, the better approximation quality achieved with the SVD model is clearly traded for more local models.

With these results one can not clearly favor one of the three domain models if the RMSE should be minimal. Each model shows best performance on at least one benchmark test or learning scenario. Observable is only a tendency that the SVD model is best suited to achieve maximal approximation quality as it has the smallest RMSE in two of three cases in both the offline and the online learning scenario.

Also, there is no clear advantage of one of the domain models if one compares the needed number of local models. In the Abalone benchmark test they are in four of five times exactly the same. Only the HLAM with the SVD model trained with the online algorithm has 20 instead of 22 models. In the Auto-Mpg benchmark test the CD model needs the most local models, while the SVD and HED model achieve similar results in

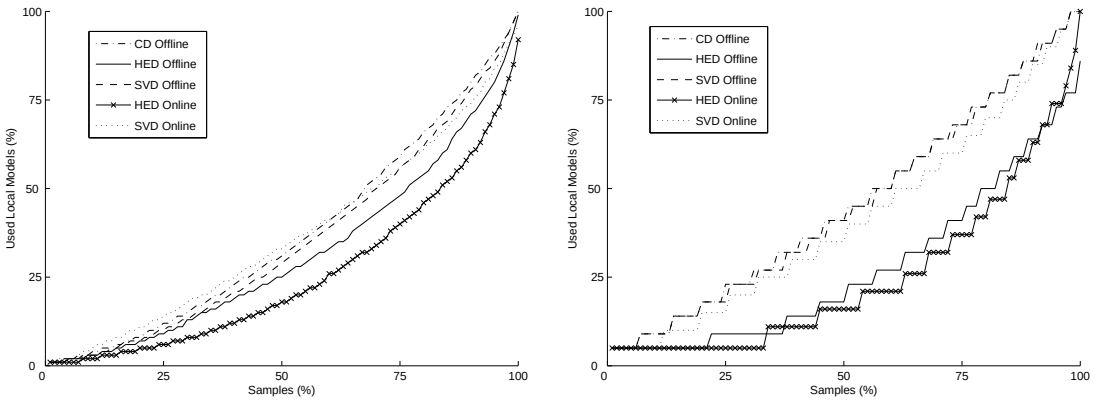


Figure 4.1.: Statistics about the usage of local models for the Mackey-Glass and Abalone benchmark test.

both the offline and online learning scenario. In order to allow a fair comparison with the Mackey-Glass data set, two extra HLAMs with the HED and SVD model are trained that got equal RMSE results as the CD model listed in Table 4.4. Using 32 local models the HLAM with the HED model achieved a RMSE of 0.0045, while 39 models are needed with the SVD model to obtain a RMSE of 0.0043. So, with 99 local models the CD model yielded again the worst result and the HED has a small advantage over the SVD model.

Usage of local models: The trained HLAMs can be analyzed w.r.t. their efficiency. One can ask how many of the established local models are really needed to approximate the test samples. It could be possible that only a small number of local models explain a large amount of the test data. In such a case the learning algorithms would have created local models that are redundant in the test phase. This question can be answered by counting how many test samples are assigned to each local models. This can be done since the exclusive gating law is applied. An answer to this question can show two different things. First of all, some conclusions about the data can be drawn. If only a relative small number of local models is needed to process many samples, the learning problem at hand can not be very complex. In the other case, it shows that the input space has to be divided on a fine scale in order to capture all the variances of the data. This data intrinsic component of a statistic about the usage of the local models can not be compensated for the three benchmark tests since the needed analytic knowledge about the learning problems is not available. Still, by comparing the different HLAMs for the same data, one can get insights about the different domain models and the two learning algorithms. This is done for the Mackey-Glass and the Abalone benchmark test.[2] The results are plotted in Figure 4.1.

[2]The Auto-Mpg data set is not used as the gained HLAMs for it contain too less local models to allow a interesting comparison.

First of all the two graphs show that eight out of the ten trained HLAMs contain no redundant local models. That indicates that the input-output relations of the two data sets have a certain complexity that can not be approximated with a very small number of linear models. The two exceptions do not alter this conviction as in one case (Mackey-Glass, HED, offline) only 8 % and in the other (Abalone, HED, online) 14 % of the local models are not used in the test phase. Another observation is more important: in all cases the percentage of used local models does not grow as fast as the percentage of tested samples, i.e. all plots are below the bisecting line $y \mapsto x$. For example 50 % of the Mackey-Glass test samples can be handled by 29 % of the local models of the offline HLAM with the SVD model. This means that some local models are more often used than others and indicates that there exist some larger regions of the input space where single linear models are enough to approximate the target function. Admittedly, this conclusion assumes that the test samples are uniformly distributed over the input space. Otherwise a single local model could be used more often than others because more samples fall into its domain by pure chance. But the finding can be supported by a more detailed analysis of one trained HLAM: the HED domains of the more frequently used local models have larger values for their scaling factors s, i.e. the resulting hyper-ellipsoids have larger diameters than those of other local models.

If such larger regions exist in the data, it is favorable if the learning algorithms would establish domains for those regions. The graphs show that the three proposed domain models facilitate this goal differently well. Some plots are more bent to the lower right corner of the figures than others. For both benchmark tests and both learning algorithms the HLAMs with the HED model have the most bent graph. The other graphs show more or less the same result. So, the HED model promotes the creation of domains that surround large regions of the input space. This is beneficial if the trained HLAM should be analyzed in order to understand how the input is mapped to the output. On the other hand it can also ruin the approximation quality since the established large domains may not fit to the data. But since the approximation quality of the HLAMs with the HED model are very competitive (cf. Table 4.4) they must be regard as the best models in this efficiency test.

Sparseness of SVD domains: In case of the SVD model, another question concerning efficiency is the number of support vectors (SV) that are needed per domain. A high number of SVs is both memory and computational demanding. While the later category has only practical implications the former gives also insights about the compactness of a built HLAM. The sparseness of the support vector domains i.e. the relation between the needed support vectors and the available training samples can be examined.

Table 4.6 lists different statistics about the relation of SVs to the training samples (TS). The results for the Mackey-Glass and Abalone benchmark test stem from the same HLAMs which performance is presented at the beginning of this subsection. Only for the Auto-Mpg results two new HLAMs are trained. This time all 320 samples of the data set are used for training so that the variance of the different training sets is of no concern. Given are three statistics: the mean number of SVs and the mean sparseness over all local

Table 4.6.: Statistics over needed support vectors (SV) in relation to training sample (TS) over all local models.

	Benchmark Test	Mean SV	Mean SV/TS	Total SV/TS
Offline	Mackey-Glass	9.43 ± 6.1	$95\,\% \pm 8\,\%$	94 %
	Abalone	121.77 ± 34.4	$90\,\% \pm 17\,\%$	89 %
	Auto-Mpg	23.00 ± 4.2	$12\,\% \pm 0\,\%$	12 %
Online	Mackey-Glass	11.00 ± 5.4	$97\,\% \pm 6\,\%$	97 %
	Abalone	138.80 ± 16.4	$97\,\% \pm 8\,\%$	97 %
	Auto-Mpg	9.08 ± 2.6	$20\,\% \pm 4\,\%$	20 %

models (together with the corresponding standard deviations) and the total sparseness of the HLAMs i.e. the total number of SVs divided by the number of all training samples.

The listed results reveal that the HLAMs for the Mackey-Glass and Abalone benchmark tests are using practically all training samples as support vectors. Only the Auto-Mpg data set allows an acceptable sparseness of below 20 %. In all cases the offline algorithm creates support vector domains that need less training samples than the online version.

The results must be seen in relationship with the used values for the meta-parameter σ of the kernel function (see Table 4.5). The kernel width for the Auto-Mpg benchmark test is 50 times larger than those of the other tests. Since the data of all three sets have a similar range in each dimension it is plausible that the domains for the Auto-Mpg data need a comparably small number of SVs. It appears that the domains for the other two tests are overfitted like exemplified with the lower right panel in Figure 3.7 on page 42. But as the value for the kernel widths are selected in order to minimize the RMSE of the whole HLAM this overfitting is no problem or might even be necessary for the best approximation of the target function. More detailed tests that examine the trade off between different kernel widths and the resulting RMSE are not conducted for this thesis. The presented results about the sparseness of SVD domains should only reveal the not necessary but possible shortcomings of this domain model.

Samples unused by the online learning algorithm: As described in Subsection 3.7.2 a shortcoming of the online learning algorithm is that training samples may not be assigned to any local model throughout the whole run-time of a system. The samples are stored in a buffer that may grow unboundedly in the proposed implementation. A valid question is how many samples are left unused in this buffer for the different benchmark tests. Table 4.7 shows the percentage of unassigned training samples. They range between 21 % and 32 %. The HED model has a small advantage over the SVD model (e.g. 21 % vs. 29 % for the Mackey-Glass data set). This amount of unused data seems quite high. These results can be interpreted in two ways: as the approximation quality of the gained online HLAMs is competitive the unused samples may represent redundant information which is negligible. But since the offline algorithm still produces better results there could be space for algorithmic improvements. These two aspects do not contradict each

Table 4.7.: Percentage of training samples that are not assigned to a local model during the online learning.

	Mackey-Glass	Abalone	Auto-Mpg
HED	21 %	29 %	25 ± 6 %
SVD	29 %	32 %	25 ± 6 %

other. Both will have effects. The only certain conclusion from this statistics is that an upper bound for the buffer size should be considered for real-world applications.

4.4. Comparing Different Gating Laws

In Section 3.2 the mixing (MG) and the exclusive gating (EG) law are proposed to define how the output of the local models should be combined to produce the final output of a HLAM. Tests are conducted to compare these two with the different benchmark tests, learning algorithms and domain models. Since the gating laws have no effect on the creation of a network the HLAMs trained for the tests of the last section can be used to calculate the results for the mixing gating law. In order to apply the mixing gating law on the CD model its activation function has to be transformed to positive values (cf. Subsection 3.2.2). To do so, the exponential function is used so that:

$$a_{CD}(x^{(D)}) = \exp(-\|p - x^{(D)}\|).$$

Table 4.8 shows the differences of the achieved RMSEs. The given values are the RMSE of the EG law subtracted by the RMSE of the MG law. Hence positive values indicate a benefit of the mixing gating law. This happens only in the tests with the HED model and is quite small. The MG law degrades the results of the CD and SVD model for all benchmark tests.

This comparison is not totally fair since the MG law is applied in HLAMs which meta-parameters are optimized w.r.t. the RMSE achieved with the EG law. It could be the case that the MG law works better for HLAMs of different configurations (e.g. more or less local models). To examine this possibility the HLAMs trained for the meta-parameter selection are validated with the mixing gating law. This is done only for the HLAMs with the HED model since these seemed to be most promising. With the selection process 40, 16, and 48 HLAMs are trained for the Mackey-Glass, Abalone and Auto-Mpg benchmark test, respectively. For these networks the differences of the RMSEs are plotted in the upper parts of Figure 4.2. In the lower parts of these graphs the number of local models of the corresponding HLAMs are drawn. The RMSE results are ordered by the number of local models.

First of all one should note that the mixing gating law performs more often better than the exclusive gating law. For all benchmark tests the mean of the RMSE differences is positive. Due to the scale of the graphs it is not visible but in some cases the benefit of the MG law is very large (Mackey-Glass: $\max \Delta\text{RMSE} = +3.2609$, Abalone: $\max \Delta\text{RMSE} =$

Table 4.8.: Comparison of the exclusive and the mixing gating law. The RMSE achieved with the exclusive gating law subtracted by the RMSE of the mixing gating law is given.

		Mackey-Glass		Abalone		Auto-Mpg	
		Offline	Online	Offline	Online	Offline	Online
ΔTest RMSE	CD	−.1857	×	−.0033	×	−.0783 (±.02)	×
	HED	+.0002	−.0043	+.0014	+.0022	+.0006 (±.01)	+.0799 (±.01)
	SVD	−.1857	−.1784	−.0037	−.0090	−.0378 (±.01)	−.0145 (±.00)

+0.2156, Auto-Mpg: $\max \Delta\text{RMSE} = +0.0676$). Still the best absolute RMSEs are achieved with the EG law. This could be grounded in the fact that the two algorithms implicitly favor the EG law due to their strict assignment of a training sample to exactly one local model (cf. Subsection 3.7.2).

The ordering of the results w.r.t. the needed local models allows to recognize a similarity between the three graphs: the stems on the left side are more often below zero than on the right side. This means that the EG law seems to be better suited for HLAMs with a comparatively small number of local models. The MG law seems to be more successful if many outputs can be mixed. A possible explanation could be that the local models of a small HLAM have to be more specialized to their domain than in a case where the differences of the local models can be smaller since more are distributed in the input space. Consequently, a mixture of adjacent models might work better if their differences are only gradual. To verify this proposition one would have to compare the local models and analyze their activation functions in order to understand how these can be mixed more successfully. As a rule of thumb one can state that it is reasonable to apply the MG law in HLAMs that have a larger number of local models and which domains are defined with the HED model. Then one may profit from the continuous output of the mixing gating law. In all other cases the EG law has clearly to be favored.

4.5. HLAMs with Different Local Models

One benefit of the HLAM approach is that the learning algorithms are not specialized to a certain machine learning technique used to train the local models. So, one can freely chose a technique that is appropriate for a specific data set. To illustrate possible effects of this decision the Mackey-Glass and Abalone benchmark tests are repeated with polynomial of order 3 and 7 as local models. These polynomials are trained with the same least squares method as the linear models of the standard configuration of a HLAM. As the rest of the configuration is left unchanged the gained networks can directly be compared w.r.t. the achieved RMSE and their number of local models with the results from Section 4.3. The results are listed in Table 4.9, while the applied meta-parameters are summarized in Table 4.10.

Different effects are observable. For the Mackey-Glass data set the RMSE and the

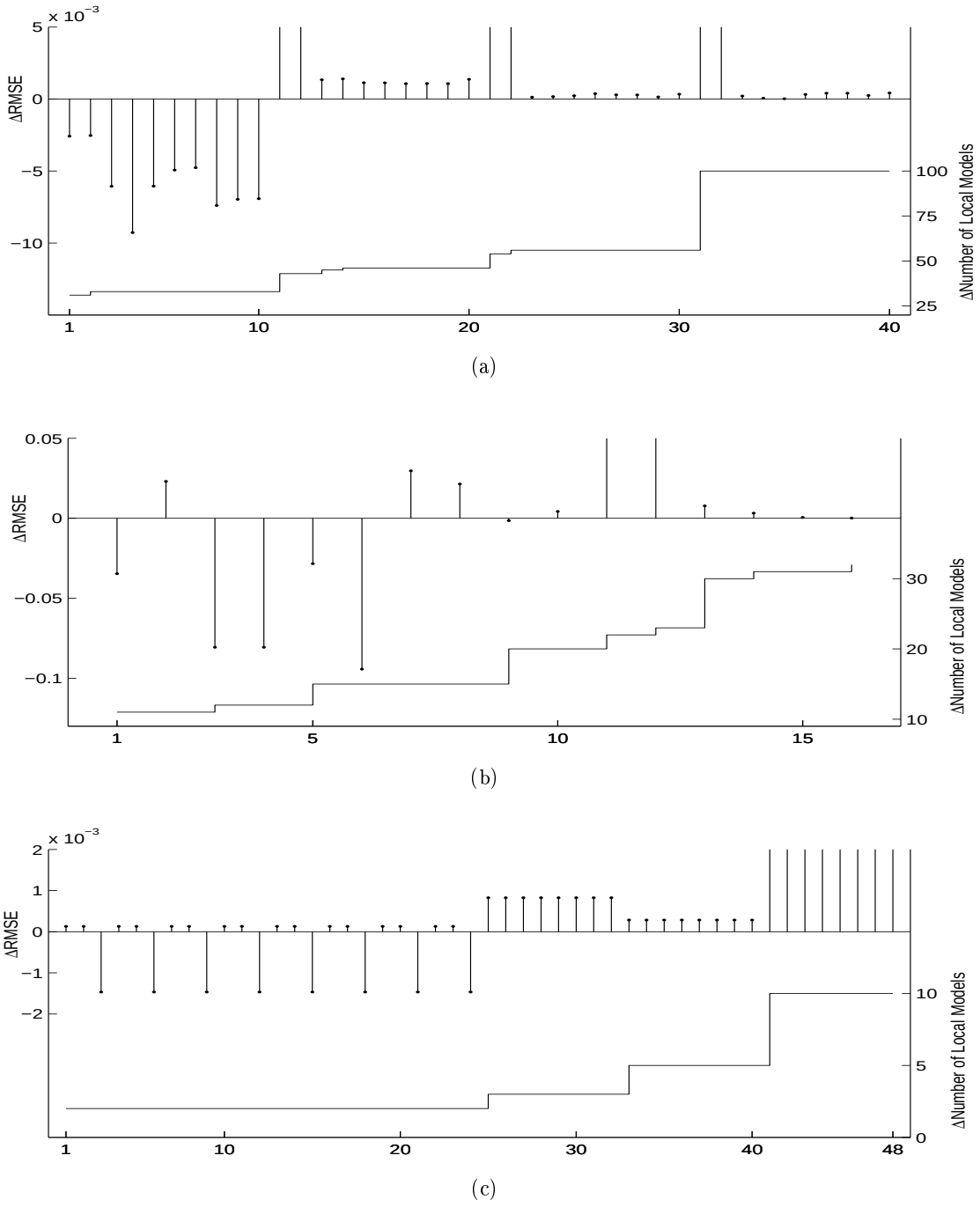


Figure 4.2.: Comparison of exclusive and mixing gating law on the Mackey-Glass, Abalone and Auto-Mpg benchmark test in panel (a), (b) and (c), respectively.

Table 4.9.: Results of different local models for two benchmark tests.

	Order of Polynomial	Mackey-Glass		Abalone	
		Test RMSE	# Models	Test RMSE	# Models
Offline	1	0.0025	92	0.0477	22
	3	0.0015	27	0.0439	2
	7	0.0089 ± 0.004	12.42 ± 0.57	$4.7 \cdot 10^9 \pm 4.1 \cdot 10^9$	3.0 ± 0.0
Online	1	0.0064	98	0.0488	22
	3	0.0038	54	0.0799	3
	7	0.0031	22	18.99	2

Table 4.10.: Values of meta-parameters of the HLAMs trained for the benchmark tests with different local models.

	Meta-Parameters	Mackey-Glass			Abalone		
		1	3	7	1	3	7
Offline	*minSamples*	6	10	50	100	1000	650
	ϵ	0.001	0.001	0.001	0.005	0.0001	0.00001
Online	*minSample*	5	10	25	100	750	1250
	maxBufferSize	50	25	100	100	1000	1500
	ϵ	0.0005	0.0001	0.0001	0.005	0.0001	0.0001

number of needed local models can be reduced considerably by increasing the order of the polynomials. With the online learning algorithm the RMSE drops to half of the original value and the number of local models is even more reduced by almost 80 %. In the offline learning scenario the polynomial of order 3 shows a very similar improvement. Only the results of the polynomial of order 7 does not fit to these observations. During the tests it became obvious that the results for the same training configuration are varying more than in other cases. This is due to the indeterministic clustering process employed in the offline learning algorithm. This process can generate different subsets for the same training data. These variances have greater impact on the determination of polynomials of higher than of lower order since more training samples are needed for their robust estimation. To get more reliable results, the tests with the offline learning algorithm and the polynomial of order 7 are repeated in 50 trials so that the mean and the standard deviation of the results can be given. They show that the number of local models can further be reduced with a polynomial of order 7 but that the approximation quality degrades clearly.

The same effect is visible with the Abalone data set: the number of local models decreases with the increasing order of the polynomials. But the achieved RMSEs are typically worst than with simple linear models. Only the offline HLAM with the polynomials of order 3 shows a small improvement. The mean results of the polynomials of order 7 are extremely bad and - when trained with the offline algorithm - equally

Table 4.11.: Results with the model validation criterion based on cross-validation.

	Order of Polynomial	Mackey-Glass		Abalone	
		Test RMSE	# of Models	Test RMSE	# of Models
Offline	1	0.0044	108	0.0457	22
	3	0.0029	40	0.0474	3
	7	0.0097 ± 0.004	14.46 ± 0.6	0.2309 ± 0.075	3.0 ± 0.0

unstable. Even the best HLAM of the 50 trials achieved only a RMSE of 0.1918.

This different outcome of the two benchmark test is certainly grounded in the fact that the Abalone data set has twice as many input dimensions than the Mackey-Glass set. Since the number of coefficients of a polynomial grows exponentially with the number of input dimensions the number of needed training samples becomes rapidly high. With a limited training set this leads of course to problems. The large demand of training sample is reflected in the meta-parameter *minSamples* (cf. Table 4.10) which have to be set to much higher values than for the two benchmark tests. This is also the reason why the number of local models must decrease with polynomials of higher order.

4.6. Comparing Different Model Validation Criteria

The offline learning algorithm creates new local models only if the approximation quality of the already trained ones is not sufficient. To determine the approximation quality two criteria are proposed in Section 3.5. In the following results are presented that are achieved with the validation criterion using cross-validation. To allow a simple comparison with the other criterion based on the training error the tests from the last section with different local models are repeated. These are especially interesting since the cross-validation is best suited to validate local models that are very flexible (cf. Subsection 3.5.2). All tests are conducted with a 5-fold cross validation.

Table 4.11 shows that the performance is degraded in five of six cases. Both the RMSE and the number of needed local models increased. Only for the Abalone data set the HLAM with polynomials of order 7 produced a better result (the minimal RMSE of the 50 trials is 0.0840). Interestingly this test has the worst outcome in the test series with the validation criterion based on the training error. Despite the fact that the achieved RMSE is still not competitive this difference of the two model validation criteria indicates the possible benefit of the cross-validation criterion. As argued in Subsection 3.5.2, it could help to improve results with local models that are very flexible. Their liability to overfitting might be restrained. Still, the results of the Mackey-Glass benchmark test do not support this conclusion that strongly. There, also the HLAM with polynomials of order 7 does not work better with the cross-validation criterion than without. But the decline in performance is clearly not as strong as for polynomials of lower order.

The baseline of these tests is that despite of its theoretical shortcomings the model validation criterion based on the training error is sufficient for practical applications.

The cross-validation criterion seem to be only helpful when very flexible local models should be used. It could be applied in data-rich situations and if a higher number of local models can be accepted.

4.7. Validation of the Algorithm to Unify Domains

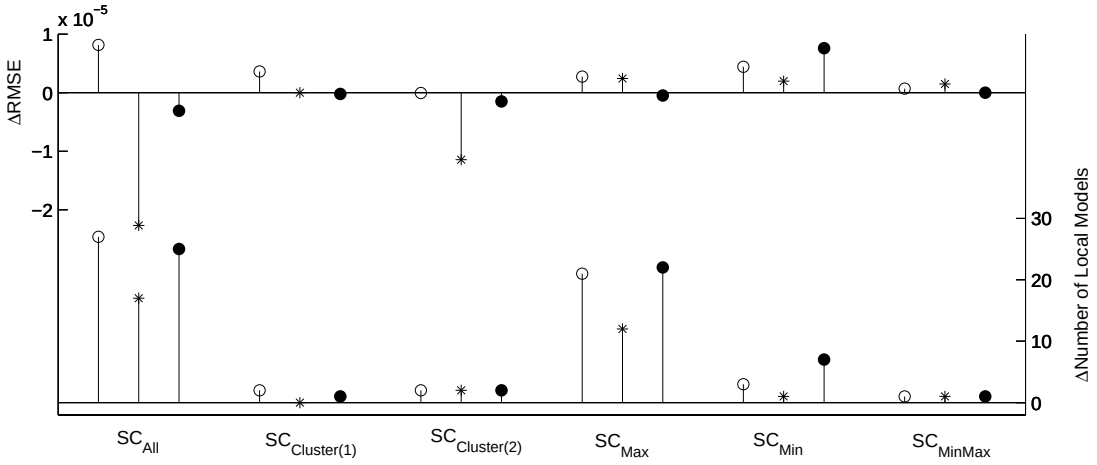
The methods proposed in Section 3.6 promise to reduce the number of local models and maybe to improve the approximation quality of a HLAM. In several tests with the Mackey-Glass data set the algorithm to unify domains and the selection criteria are validated. This benchmark test is chosen as the HLAMs trained for it contained the most local models and hence the possible benefit should be most obvious. Consequently the unification algorithm is applied on the five HLAMs trained with the offline and online learning algorithm and different domain models as specified in Section 4.2. In all cases the domains are unified by association.

The results of the different selection criteria for the three domain models and two learning algorithms are plotted in Figure 4.3. Given are the differences of the achieved RMSEs and the number of local models. The results of the original HLAMs are subtracted by the results of the HLAMs after the unification algorithm are applied to them. The RMSEs of the three domain models are scaled in order to be plotted appropriately. The used factors are noted in the caption of the figure.

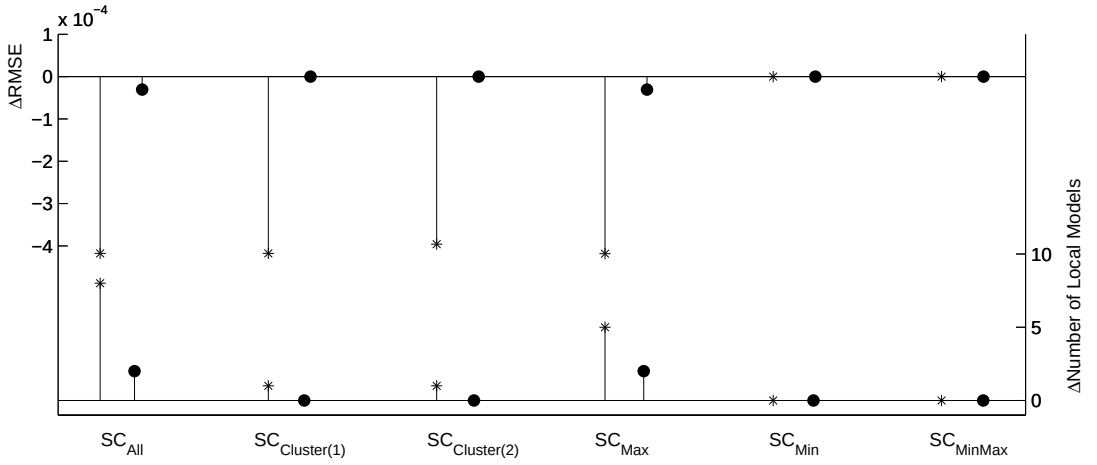
Altogether the plots show that the proposed algorithm successfully unifies domains without a significant effect onto the approximation quality. The different selection criteria help to reduce the number of local models differently. The SC_{All} and SC_{Max} criteria show the best results. With these criteria the size of the HLAMs trained with the offline and online learning algorithm is diminished by up to 27 % and 8 %, respectively. The other criteria performed only poorly for all tests. Regarding all offline trained networks the most positive effects could be obtained with the HLAM with the CD model: the number of local models is reduced significantly and the approximation quality is enhanced. For the online HLAMs the results are not equally promising. The highest reduction by eight hyper-elliptical domains is traded for an increase of the RMSE by 0.0004. On the HLAM with the SVD model the unification algorithm shows practically no effect.

4.8. Summary

The experiments described in this chapter prove that the HLAM approach can be successfully employed as a machine learning technique. Its quality of approximation achieved on one synthetic and two real-world data sets outperforms eight different competitors. These results could be produced with several differently configured HLAMs, i.e. HLAMs that use e.g. different domain or local models. This confirms that the options proposed for the various aspects of a HLAM are useful. With the series of experiments the qualities – their advantages as well as their disadvantages – of the new learning algorithms and models could be examined.



(a) Results of offline experiments



(b) Results of online experiments

Figure 4.3.: Comparison of the different selection criteria for the domain unification algorithm. Panel (a) and (b) shows the outcomes of the offline and online HLAM, respectively. The results are shown so that positive values indicate the benefit of the unification algorithm. In panel (a) the ΔRMSE results of the CD and SVD model are scaled with the factor 0.01 and 0.1, respectively. In panel (b) the factor for the SVD model is 1000.

Both the offline and online learning algorithm for the HLAM approach produce networks that achieve best results on the benchmark tests. As expected the offline algorithm realizes better approximation quality than the online version. The variances in the training results of the offline algorithm induced by its indeterministic clustering method seem to be negligible in practice. The analysis of trained HLAMs revealed that the amount of training data that remains unused by the online algorithm is quite high. This indicates that in practice an upper boundary for the buffer size that handles these data is recommended. The experiments with the three domain models document that the possible outcomes are comparable good. W.r.t. the obtained approximation quality the SVD model offers slight advantages, while the CD model certainly requires the most local models. For only one data sets the sparseness of the SVD model are acceptable. In the other two cases practically the whole training set has to be employed as support vectors in the trained HLAMs. This might be the result of an inappropriate choice of the kernel width. The exclusive gating law performs generally better than the mixing gating law. Still, for HLAMs with a relative larger number of local models the MG law is applicable if the continuous output of the MG law is really needed. Experiments with local models of different flexibility demonstrate two expected effects: with polynomials of higher order the approximation quality can be improved and the number of needed local models can be significantly reduced. But the order can not be increased arbitrarily as the parameter estimation becomes unstable. In such cases the proposed model validation criterion based on cross validation can be helpful. Otherwise the criterion grounded on the training error proved to be sufficient. Finally, the algorithm to unify domains showed good results with the selection criteria SC_{All} and SC_{Max}. The reduction of local models is most promising for HLAMs trained with the offline learning algorithm and the CD model. On HLAMs established with the online version the performance is not as good.

After all one can favor the following configuration for a HLAM: the HED model should be used with local models that are either linear models or a polynomial of a rather small order. The exclusive gating law should be applied and the model validation criterion should based on the training error. The algorithm to unify domains can be invoked with the SC_{Max} selection criterion. This choice promises to produce best results with the HLAM approach w.r.t. both the approximation quality and the number of local models. It also restricts the computational burden to a minimum*:* the HED model has less parameters than the SVD model and its estimation does not require an extra meta-parameter optimization. The model validation criterion is the simplest and the unification algorithm using the SC_{Max} selection criterion tests only a subset of the candidates of the SC_{All} criterion.

Part II.

HLAM for a Visual Robotic System

Chapter 5

Learning-based Visual Robotic Systems

The second part of this thesis is concerned with the application of the HLAM approach in a robot vision system. In this chapter the scientific background of such a robot vision system is outlined. In the first section general goals and problems are described along with a brief history of the major directions for appropriate software architectures. The second section highlights the most important aspects how machine learning techniques can be employed in robotic systems. In accordance with the scope of this thesis the emphasis is laid on learning to generate actions rather than learning for perception. Both sections are written as introductions to the very broad research field of robotics. For more detailed descriptions, the reader is referred to the given citations.

5.1. Goals and Problems of Robot Vision Systems

As Erdmann [35] concisely stated, the ultimate goal in engineering of robot systems is: "We want specific things to happen quickly and cheaply." As the Czech word 'robot' implies we human want that artificial systems act as workers for us. Robots[1] should take over all these jobs which are physically too demanding, too exhausting or just too tedious for us. They should work in factories, provide help in our everyday life or replace us in environments that are hazardous. In each application area robots have to perform specific tasks in an efficient manner. So, the expenses of resources for robots should be well balanced with the gained benefit in product quality or in comfort of our life. A very important component of this economical constraint is the needed effort to develop robot systems because – simply stated – the demanded tasks can not easily be solved. The plain reason for that is, the world we are situated in is very complex. This property of the world is grounded in the huge variations of its state: everything is constantly changing at different time scales. Hence acting in the world depends on an adequate perception of its state.

The thorough exploration of foreign planets or inhospitable environments on the earth requires highly autonomous systems that can work in unforeseen circumstances. Household assistant robots have to cope with the tremendous and rapid variations of situations. Even in such highly customized environments for assembly tasks things still go

[1] For a proper definition of a robot system see e.g. [85].

wrong. Robots that should make specific things happen quickly and cheaply require three non-trivial components: appropriate effectors, instructive sensors and powerful working strategies. Of course, effectors have to have the right size, dexterity and strength to handle things or to move a robot. Sensors must acquire a system internal picture of the world that serves as a valid base to realize the desired activity. The interplay between action and perception has to be well organized so that to every possible situation an appropriate response is computed.

The variety, power and precision of available hardware for effectors and sensors is highly developed. Industrial arm robots can move hundreds of kg at high speed to position within sub-millimeter accuracy. Laser scanner and video cameras can acquire sensory data of the world that is far beyond any precision humans or animals can obtain. Still, even the smallest insects demonstrated competences and adaptivity in natural environments that are impressive when compared with common artificial systems. So, the shortcoming of current robotic systems lays clearly in the field of software.

Two major problems are limiting robots to realize complex and goal-driven actions. The purposive interpretation of sensory data remains a hard problem. Software modules have to bridge the big gap between the available low-level signals and the needed instructive information to control the effectors. Especially the interpretation of image data has not been accomplished with generic methods. The existing ones that are sufficient fast and robust are highly customized (the methods or/and the environment). But cameras are very important long-range sensors because they are very mature techniques that can easily provide huge amount of information.[2] Hence this discussion emphasizes robot *vision* systems. The second problem is that no generic software architecture exists that defines how effectors can be combined with appropriate sensors in order to solve real-world problems. No program has yet been developed that really keeps the promises of the famous "General Problem Solver" of Newell and Simon [81]. In the following the two problems are described in more details.

Data Processing in Robot Vision Systems The processing of sensory data is not just a problem of computational power although e.g. real-time color image processing with quite simple convolution filters is still challenging. The actual problem is to extract from the stream of sensory data these information that are important to solve the robotic task. The interpretation of data must tightly be coupled with goals given by the system user in order to control the effectors appropriately. The task dependence of a robot is the reason for the strong customization of its data processing methods. Moving through a corridor in order to search for a specific item can be solved with quite different scene analysis mechanisms than moving through it in order to clean the corridor.

Another major problem to realize the mapping from perceptions to actions is the difference in data representation. The output of sensors is first of all given as high-dimensional continuous data streams. Processing techniques can transform these into more invariant

[2] Another reason for the popularity of cameras in robotics is that humans rely so strongly on their sight. It is a natural bias to that kind of sensor and it significantly helps to debug robotic systems since the modality to perceive the world is the same.

but still continuous signals. Current machine vision approaches [31, 38] mainly combine explicit models of geometric entities with statistical methods in order to reach more abstract descriptions of observed scenes. Secondly, goals that express the intentions of the human user or designer of the system are most efficiently communicated in natural language or with gestures. In the classical artificial intelligence (AI) approaches [22] goals are represented as symbols (i.e. discrete numbers) and computational reasoning with them is grounded in a suitable logic (e.g. first order predicate or temporal logic). Finally, actions of effectors are typically defined w.r.t. a robot specific coordinate system. Each point in this coordinate system corresponds to a certain configuration of the effectors. Movements of the robot are represented as continual trajectories in this coordinate system. These three forms of representation were developed in separate research areas and do not easily fit together. Not before the late 1980s serious attempts were made to combine the results to create powerful robot vision systems. Despite the spent effort, the scientific community is still working on better representations and means to combine them (e.g. [104] proposes an unified algebraic framework).

Important to note about the data processing in a robot vision system is the need of of abstraction and concretization. On one hand side, noisy low-level signals have to be stripped of all the components that are irrelevant to the task at hand. The signals have to be transformed so that invariants in the data can be compared by some means to high-level goals of a system. On the other hand, these abstract goals have to be translated into concrete motor commands that precisely drive the effectors.

Software architectures for Robot Vision Systems This chain of data processing has to be reflected in the system's software architecture. But there exist quite different ideas to solve this. Classical AI approaches delineate three functional modules: a sensing, planning and acting component [82]. The first module generates an internal model of the world which is matched by the planning module against a given goal state. In case of a disagreement a symbol-based reasoning mechanism has to devise a plan which is finally executed by the acting component. Characteristic for such systems is their strong emphasis on symbol processing which implies extensive modeling of the world and an unidirectional linear data stream from the sensors through the planning component to the effectors.

Tests outside of customized environments showed that this approach is not practical. The world is too complex to be manually modeled. Open-loop plan execution turned out to be inadequate to realize system capable of fast reactions. So, it became obvious that abstract reasoning is not as important for acting in the world as assumed. As opposed to the strict three module structure Minsky described in [77] a *society of agents* that is a set of single software modules that are connected over various data channels. These agents are organized according to both cooperative and competing principles. Such a society of agents is supposed to solve problems in a better way than the programmer had in mind when designing each single part. Complexity should emerge from the combination of simple parts.

Although Minsky was much more interested in general theories for cognition than

actual robot systems his ideas resembles these of Brooks who proposed the afterwards so-called behavioral design approach with his famous *subsumption architecture* [15]. Brooks and his numerous successors argue that complex behaviors[3] of robots can be produced by a loosely coupled network of simple asynchronous processors that have full access to both sensors and effectors. The full access is necessary to realize fast feedback cycles that involve perceptions and actions. They follow a strict bottom-up design approach which emphasizes low-level reactivity instead of high-level planning. The goal was to realize systems that are adaptive to their environment because a new insight was that intelligent behavior of an acting entity is a product of the entity *and* its environment. The argumentation is that quite simple mechanisms in robots can make specific things happen as long as these robots fit to their environment. A simple example for this problem solving strategy is the combination of more or less random moves with basic obstacle avoidance methods that drive a mobile robot like the commercially available "Roomba" (see 'www.irobot.com') through rooms in order to vacuum-clean them.

This design paradigm can be seen as the antithesis to symbol-based AI. Brooks itself stresses this rivalry with articles entitled "Elephants don't play chess" [16] and "Intelligence without reason" [14]. But his key concepts of *Situatedness*, *Embodiment*, *Intelligence* and *Emergence* [14] (see [104, 85] for reformulations) lead to a serious problem: the system designer is quite limited to introduce high-level goals into the system. The architecture is not meant to contain a centralized module that plans and pursues long-term goals. So, the designer can not easily integrate knowledge that would be gained by a normal engineering processes where goals are decomposed into subgoals. Instead one is supposed to define only basic behaviors from which the intended overall behavior has to emerge. But it remains unclear what kind of generic design principles can be applied to develop behavioral robots that do what they are intended to do (for detailed criticism see [110, 23, 85]).

The synthesis of these two research directions has became present in robot vision systems. In 1991 three research groups [29, 44, 9] independently developed quite similar architectures with three layers that combines purely reactive with long-term planning modules (for an overview see [45]). An intermediate layer is used to keep the balance between these two extremes. This architecture is still relevant as it was applied e.g. in the robot that won the first time the DARPA Grand Challenge [109].

Another substantial shift in methodology is the replacement of intensive modeling by the appliance of learning strategies. Two reasons are important: explicit modeling of the world is very time-demanding and the results tend to be too inflexible. The advances in machine learning techniques made is possible to teach different aspects of robotic tasks so that the system generalizes over the given training samples. Furthermore, MLTs can help to realize modules (especially those directly connected to sensors) that are adaptive to changes in the environment. So, learning strategies became very important to ease the development and increase the robustness of robot vision systems.

[3] In this thesis the view of Pauli [85] is taken where instructions and behaviors are differentiated. The latter class of robot movement always implies a feedback cycle, while instructions are commanded blindly to the effectors.

5.2. Learning in Robot Vision Systems

After clarifying the application area and delineating first reasons, more aspects of machine learning techniques in robot vision systems are discussed in this section. The presented aspects are selected w.r.t. the topics of this thesis which are concerned with learning of robot movements. For a more general discussion see [85] or [108].

Picking up the ideas of the behavioral approach, the development of a robot vision system should emphasis a minimalism principle. As much competences should be achieved with as little expenses. Certainly, machine learning techniques have great potential to promote this principle. All kinds of learning paradigms were applied in robotic systems: e.g. genetic algorithms in [67], reinforcement learning in [51], supervised methods in [85]. These were used to realize various aspects within a system on different levels of abstraction and time scales.

Two task fields can most clearly be separated: learning aspects of perception and those of acting. Various MLTs are used for such problems like image segmentation, object recognition or categorization (for examples see [18, 69, 86]). Obvious, for these problems different types of data have to be processed. They can be very low-level continuous signals or rather high-level object class labels. For an overview about learning-based perception see [31]. Of more interest for this thesis are learning strategies that generate actions of a robot.

Learning Actions The typical task is to learn a transfer function that maps directly from sensor data to motor commands which steer the effectors. Such transfer functions define basic behaviors[4] like e.g. wall-following or obstacle avoidance since they are used to realize close-loop controllers. These are meant to perform fast perception-action cycles that are necessary for highly reactive systems.

These transfer functions are classically hand-crafted by means of explicit models. But they can also be trained with different learning strategies. E.g. in [85] and [55] an arm robot performs certain movements and the resulting changes of sensory data (i.e. image position of a tracked object) are measured. By this means samples can be collected how robot commands correspond to sensory data. Given a set of such samples a supervised learning technique is used to establish the needed transfer function. In [98] other examples are outlined (e.g. imitation of trajectories with reinforcement learning).

Although in robotics the concept of transfer functions is mainly associated with low-level controller, it is quite general. These functions can be used at various levels within a software architecture working on differently abstract data. E.g. instead of pure image data, classified object features could be the input of a transfer function and its output could be the invocation of one basic behavior. This type of task is typically solved with manually designed finite state automatons [45]. These have the disadvantage that they are fixed. In [24] a framework is proposed that learns such transfer functions with first order predicate logic. An alternative is described in [36] where the dynamic field

[4] In the literature various terms for basic behaviors exists: movement primitives, motor skills, or primitive behaviors.

formalization of Amari [4] is chosen to model pattern formations in neuronal tissue. In both cases the correct sequence of basic behaviors had to be learned by observing a human instructor. The set of available basic behaviors were comprising such actions like "move backwards", "move left" or "grab with full grip".

Furthermore transfer functions offer a good possibility to introduce high-level goals into low-level controller. The realized basic behavior can be conditioned by decisions made at different levels within the software architecture. To do so, internal state variables can be added to the sensory data that serve as input for the transfer function. These state variables can reflect the past or the desired future of the system in order to alternate the actions of the robot. How this can be learned is described in Section 7.4.

Learning by Visual Demonstration and Imitation After outlining what should be learned by a robot vision system one can tackle the question how it is typically done and what principles are useful to be applied. Two learning strategies are relevant to the system described in the next chapter: learning by visual demonstration and learning by imitation. The first term stems from [85], while the second one is used like in [7].

Learning by visual demonstration comprises two major principles. On one hand, it stipulates that only those information of the environment are extracted from images that are absolutely relevant to the robotic task, i.e. the most basic image features should be used for guiding a robot. This minimalism principle is motivated biologically in [65] and should reduce faulty or expensive data acquisition. Furthermore, learning by visual demonstration demands that certain abilities of the system are acquired under the same circumstances as these are needed in real applications. This could mean for instance that target positions of the robot end-effector or some objects are demonstrated to the system as these would occur during its operation. Typically supervised learning techniques are used to generalize from such examples to new circumstances. As desired, by this means explicit modeling can be omitted. It offers a designer the opportunity to interactively train his or her system during run-time. Of course this remains only an advantage if the acquisition of samples is realized in a reasonable efficient manner.

Learning by imitation has a larger scope than learning by visual demonstration. It implies that the robot system observes the actions of a (human) teacher and is able to repeat these. Traditionally, this form of teaching a robot is called *programming by demonstration*. Following the current research trend, in this thesis the term imitation is used in order to increase the goals of this learning paradigm. Because imitation can mean quite different things. In the most basic case a movement would exactly be copied which requires that the teacher and the robot have the same embodiment. A relaxation would demand that the trajectory of a certain part of the teacher and the robot (e.g. hand and end-effector) are equal or at least similar. Another level of learning capability would be a system that can repeat the right sequence of single actions shown by a teacher. A further generalization would expect the robot vision system to transfer the observed actions from the specific situation the teacher was in to its own situation. E.g. when the teacher has shown how to align a pen next to a piece of paper, a robot may align a fork next to plate just because no other similar object was within its reach.

The last examples makes clear that correct imitation can require high-level cognitive abilities. The learning problem can be shifted from "observing and repeating actions" to "understanding intentions and acting appropriately to the own situation". Not without reasons the engineering community that works on robotic imitation has found assistance in the field of neuroscience, developmental psychology and social science. In a special issue [7] of the "Robotics and Autonomous Systems" journal various aspects of learning by imitation are discussed. It comprises articles about topics such as replicating trajectories [3, 83], inferring the intentions of a teacher [36] or using social cues to increase knowledge about a given situation [10]. In [6] a good introduction to robot imitation along with references to developmental psychology is given.

Learning on Different Time Scales Finally, one can ask at which point of time should learning in a robot vision system happen. In [105] three major time scales are suggested: the *ontogenetic*, the *refinement* and the *situated* time scale. In the first phase, offline learning methods should be used in order to bootstrap basic competences without the system would not be able to do anything. This phase is also helpful if the learning problem is too complex to be handled during the operation of the system (e.g. because the training sample acquisition is too time demanding). On the refinement scale, the basic competences can be improved during their actual operation by means of online learning principles. Typical examples are rather slowly changing calibration problems such as color changes over the day. The fastest learning mechanisms are needed for the situated time scale. Ultimately, the system should be capable of learning from single examples like the demonstration of a perfect swing with a golf club. For this task only methods of learning by imitation can be considered.

5.3. Summary

In this chapter various aspects of robot vision systems are introduced. Some examples were given why robots should work for us humans. Since the world in which these robot should act is so complex it was concluded that they require appropriate effectors, instructive sensors and powerful working strategies. Two main problems were identified that prevent a major breakthrough in the development of software for robotic systems. On one hand, not enough generic methods exist to realize the task-dependent processing of sensory data that are sufficiently robust and fast. Furthermore, the data representation typical for different levels of abstraction are quite incompatible and hinder a simple combination. On the other hand, there is no common agreement on a general software architecture for robot vision systems. Classical AI approaches suggested a structure where single modules are concerned with sensing, planning and executing. But the very high demand of explicit modeling of world phenomena and the typical slow reaction time disqualified such systems. As an antagonism, the behavioral approach emphasized a strict bottom-up design approach which resulted in fast reactive systems that were adaptive to their environment. The hope was that unforeseen complex behavior would emerge from specially designed basic behaviors. But it became evident that with this

approach it is very hard to realize systems that solve desired tasks which need long-term reasoning. As a synthesis of these two schools of system design the currently used architectures combine reactive with planning modules. Furthermore, they more and more rely on machine learning techniques which are identified as powerful tools to ease the development and increase the robustness of robot vision systems.

MLT are applied in two major application areas within robotics: learning perception and learning actions. The former is concerned with more robust processing of sensory data, while the latter deals with transfer functions that map perceptions to appropriate actions. Transfer functions can be situated at various levels within a system architecture. At the lower levels MLTs replace hand-crafted controller models, while on higher levels they are alternatives to fixed finite state automatons. In both cases the MLTs can be trained by following a visual demonstration or imitation learning strategy. The former relies on supervised learning techniques in order to teach a system how to react by demonstrating situations as they could occur during run-time. In this context, a minimalist design principle stipulates that only this information should be extracted from sensory data that is actually needed for desired robotic task. Learning by imitation has an even broader scope: it demands that a robot vision system is able to observe and repeat actions of a (human) teacher. The goals of this learning paradigm range from replicating trajectories to inferring the intention of a teacher. A solution to such problem would significantly ease the development of robot systems as it could be trained more human like. Finally, the three different time scales of ontogenetic, refinement and the situated learning were introduced. They refer to an offline learning phase where basic abilities are taught, to the rather slowly improvement of these abilities during run-time and the fast acquisition of new competences with learning by imitation strategies.

Chapter 6

The COSPAL System

Over the period of three years the European Union funded a project [30] with the title "Cognitive Systems using Perception-Action Learning" (COSPAL). This project was proposed and conducted by a consortium of four partners, namely the Computer Vision Laboratory of the Linköping University (LiU), the Cognitive Systems Group of the Christian-Albrechts-University of Kiel (CAU), Centre for Vision, Speech, & Signal Processing of the University of Surrey (UniS) and the Center for Machine Perception of the Czech Technical University in Prague (CTU).

The methods described in this thesis are specifically developed for the COSPAL system. To be able to describe how the HLAM approach is integrated into the system, this chapter explains the aims of the project, the developed software architecture and the concrete task of the system. These presentations are most detailed for those parts important to the application of the HLAM. Descriptions of the results accomplished by other partners can be found with the given citations.

6.1. Goals of the COSPAL Project

The COSPAL project aims to develop a new system architecture and new learning strategies for artificial cognitive systems. On the basis of a robot vision system, new forms of "interaction of continuous *and* symbolic perception and action" were studied which should result "in robust and stable motor and sensory capabilities of the system and allows a purposive behaviour" [49]. The goal was to built a system capable of incrementally learning to solve tasks. Its architecture should be structured as a multi-level network which is initially established during a bootstrapping phase and afterwards refined to a specific task (i.e. environment).

The consortium of COSPAL partners addressed – as cited from [49] – various sub-goals:

- mimicking human movements to achieve an optimal manipulator control,
- investigation of heterogeneous and homogeneous multi-level network structures for associating percepts and actions,
- development of methods for validateal learning,

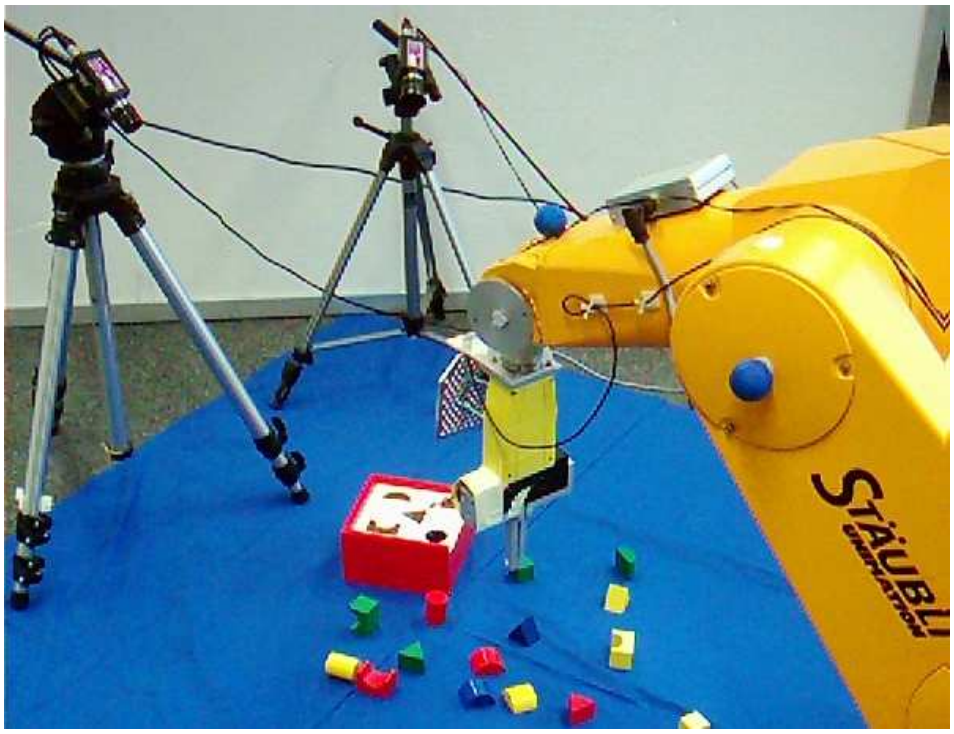


Figure 6.1.: Sample setup of COSPAL robot vision system. Two static cameras are observing the robot and shape-sorter puzzle.

- development of methods for reinforcing context dependent purposes to the network after the bootstrapping phase,
- investigation of interfaces between associative networks and symbolic layers, and
- development of suitable symbolic methods for supervising network states.

To demonstrate all these abilities, a shape-sorter puzzle was chosen as a concrete task. An exemplary setup of the system is shown in Figure 6.1. Besides the industrial arm-robot and two cameras, the shape-sorter box and the puzzle blocks are visible. The goal is that the robot learns to insert the blocks into the correct holes of the box. How the COSPAL demonstrator looks like in detail is described in Section 6.3 by clarifying its physical constraints and the dynamics of the robot.

6.2. The COSPAL Software Architecture

This section outlines the software that is realized in the COSPAL project. First, its general structure is given. W.r.t. that information the applied design principles can then be explained. In the last two subsections, the responsibilities of the research groups for the different parts of the system and their results are delineated. This way of describing

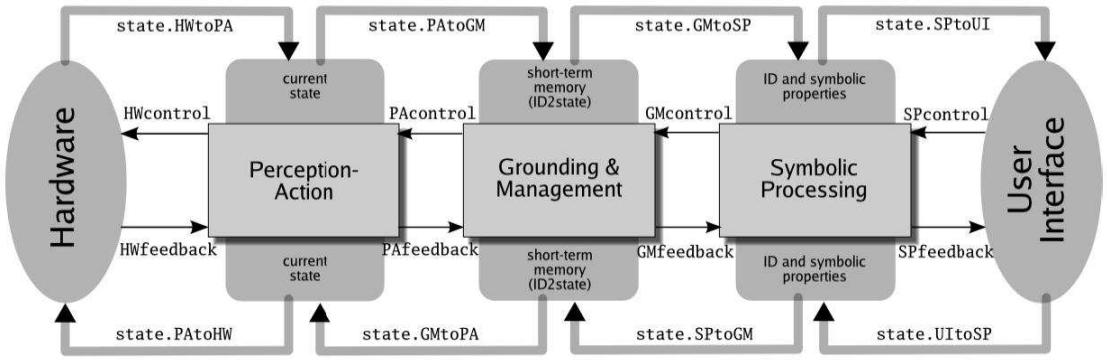


Figure 6.2.: COSPAL system architecture with the three main modules. The hardware comprises the robot and the cameras as interface to the world. The user interface is essentially a console where commands can be entered. The arrows with the thick lines represent data streams between the modules. The conveyed data becomes more and more abstract from left to right. The arrows with the thin lines represent the invocation of functionalities the different modules offer. Source: COSPAL internal paper made by LiU.

the realized COSPAL system is chosen because its software architecture is composed of functional blocks which are developed by individual partners.

6.2.1. Overview of the Software Structure

At the global scale, the static structure of the COSPAL software architecture (see Figure 6.2) contains three modules[1]: the perception-action (PA), the grounding-management (GM) and the symbol processing module (SP). As the names should indicate, the modules encapsulate functionalities that are working on data of different degrees of abstraction.

Basically, the PA module realizes the competences at the lowest level where tasks such as image processing, object recognition and basic robot behaviors are solved. The goal of the GM modules is to allow data exchanges between the PA and the SP module. This means that – in the direction from PA to SP – continuous signals are classified in order to ground symbols. The other way round, the GM module resolves symbolic IDs that refer to entities in the world to their non-symbolic representation. The SP module is responsible to fulfill the long-term goals of the system and offers a high-level interface to the user.

Besides the type of processed data, the three modules can also be characterized w.r.t. their scope of time and knowledge about the world. As already hinted, the SP module works on the global scale: it establishes and maintains plans to solve the overall task to play the puzzle. To do so, it builds the most complete and most abstract model of the world as it is perceived by the system. On the other extreme, the PA module is concerned with short-term goals that are realized with a small but detailed portion of all

[1] Note the similarity of this structure to the three layer architecture outlined in the last chapter.

sensory data. At the intermediate level, the GM module connects these two perspectives onto the world by means of a so-called short-term memory.

6.2.2. Design Principles

Characteristic to the COSPAL approach to robot vision systems is the strong emphasis on learning. The goal was to create a system that can learn at all levels of its architecture. The competences should be acquired, refined or extended at all three time scales (cf. page 91). The learning strategies vary over the three modules. As already mentioned, the used methods differ w.r.t. the type of data they are processing. In accordance to the type and the demanded task, a number of learning paradigms is employed. Supervised and reinforcement learning are mostly applied in the PA and GM module, while a search algorithm solves the task of symbol processing. The latter has the distinct feature that the system is supposed to learn by exploration. The idea is that random invocations of already established competences can lead to new abilities. Assumed is that success or failure feedback is given by either the human user or the environment (delivered by the other modules).

An important feature of the COSPAL system is the use of layered feedback cycles. To give a simple example: whenever the SP module commands a basic behavior (e.g. the grabbing of a certain puzzle block) it will finally receive a high-level feedback about the outcome of the action. This feedback is generated by similar processes started at other levels of the architecture. The difference of these processes lays in the type of exchanged data and the speed of the feedback cycles. The fastest are running at the lowest level in the PA module where the robot will be commanded to drive a distance of some millimeters and the feedback will be extracted from a new acquired camera image of the changed scene.

6.2.3. Responsibilities of the Different Project Partners

The different functionalities encapsulated by these modules were developed by the different COSPAL partners according to their expertise and research interest. The responsibilities were distributed as follow: LiU studied new methods that are concerned with problems of the PA module (e.g. image feature extraction, object recognition, robot control schemes). Although with other approaches, similar tasks were tackled by the CAU. Only one significant difference should be mentioned. The image processing methods of the CAU can establish a symbolic description (i.e. by color and shape classes) of recognized object. The GM module was the work field of CTU. They developed special banks of classifier to ground symbols needed by the SP module. Finally, UniS were responsible to implement symbol-based learning strategies that had to process the user input in order to learn the high-level goals of the puzzle game. As an additional feature of the system, CAU realized a recognizer of human movements. It solves two tasks: it can recognize a set of human movements so that a user can teach the system what has to be done in a certain situation by just showing it. The second goal was to let the robot imitate the demonstrated trajectories.

6.2.4. Outline of Applied Methods

Since a detailed review of the used techniques is beyond the scope of this thesis the reader is referred to the following selected publications. Granlund outlined in [48] and [50] the abstract philosophy that guided the work from LiU. In accordance to this, a preliminary version of LiU's PA module is proposed in [39] that worked with a simulated shape sorter puzzle. The approach follows the reinforcement learning paradigm to establish a control scheme for the simulated three DOF robot. The object recognition task is solved with an online supervised learning technique [63] that processed so-called channel encoded image features [47].

The CAU based its object recognition and categorization method on a novel incremental classifier [90, 91]. It is used to segment images (exploiting the fact that all puzzle blocks are of uniform color) and to classify contours of the found color blobs. The robot control tasks are realized with the HLAM approach as described in details in the next chapter. The human movement recognizer comprises two distinct layers. The first tracks the hand and the arm of a user performing certain gestures typical for the shape-sorter puzzle (e.g. grabbing or inserting). The gained trajectories are approximated with a PCA method so that a dynamic cell structure network can generalize to unseen human-like movements [1, 2]. To drive the robot along such trajectories, the second layer of the recognizer is trained by means of a reinforcement scheme. It learns to map the image coordinates of the desired trajectory into the robot coordinate system.

Different methods were studied by CTU to solve the symbol grounding problem in the GM module. Progress is made in the theory of Markov random field concerning the computation of their maximum posterior configuration [113]. Support vector machine learning is extended to cope with structured output spaces and complex loss functions. This research led to a new learning algorithm that can handle a huge number of linear constraints [40, 41]. The needed short-term memory to realize the data flow from the SP to the PA module could be solved with a rather simple look-up table.

Based on former research results [70], at UniS a special annotated relational graph is developed that represents scenes perceived during the puzzle games. A vertex of such a graph symbolizes one relevant object (puzzle block or hole in the box) and an edge defines the relative positions between two connected objects. During a learning phase the human user has to play the puzzle which results in the acquisition of a database of such graphs. To each graph (i.e. observed scene) the action performed by the user is associated. By this means, the database contains the information how the puzzle game is played correctly. This information is retrieved by a specialized search algorithm that is able to generalize over the database in order to chose appropriate actions in new scenes [34]. To do so, a new metric had to be realized [84].

6.3. A COSPAL Demonstrator

In this section the setup of a robot vision system and its environment is specified for the shape-sorter puzzle. The robot system is built up at the CAU and customized to the version of the PA module created by the Kiel group. First, the static properties of the

system are given, before the possible movements of the robot can be described. Both is needed to understand the implementations explained in the next chapter that realize the robot control parts of the system.

6.3.1. Static Properties

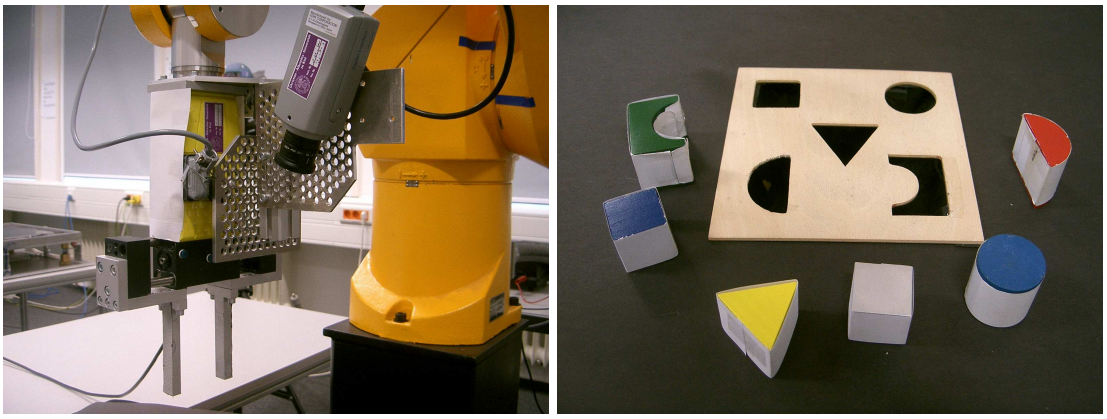
The following enumeration specifies the setup of the implemented robot vision system and the physical constraints of the shape-sorter puzzle:

- The central part of the system is a Stäubli RX90 industrial arm robot with six DOFs. Its end-effector is equipped with a gripper with two parallel bars as fingers (see Panel (a) in Figure 6.3). The gripper can only be closed or opened. No intermediate position is possible.

- Two color video cameras (Sony DFW X710) are used to observe the scene. One (focal length 6.5 mm) is attached to the end-effector of the robot. This so-called end-effector camera is slightly tilted so that both the gripper and everything underneath it can be monitored (see Panel (a) in Figure 6.3). The second camera (focal length 6.0 mm) is mounted on a tripod placed beside the robot. The images of this so-called static camera always contain the full work space of the robot and its end-effector.
  The two cameras are setup so that everything needed is visible in their images. No other arrangements are made (e.g. the optical axis of the end-effector camera is not aligned with gripper). Hence, the exact spacial relations between the coordinate systems of the cameras and of the robot are unknown.

- The working area (i.e. the ground where puzzle blocks could be placed within reach of the robot) is planar with a size of approximately $25\,\mathrm{cm} \times 35\,\mathrm{cm}$. It is virtually parallel to the x-y-plane of the robot coordinate system. It is made of polystyrene which is covered with black cardboard.

- The puzzle game comprises a number of blocks and a plate with five matching holes. The blocks are of uniform color (red, green, yellow, blue, or silver) and have either the form of a cylinder, a bridge, a triangle, a cube, or a crescent (see Panel (b) in Figure 6.3). In the following both blocks and holes are referred to as objects.
  The plate with holes is originally the cover of the shape-sorter box, but it is placed directly onto the working area for the experiments. To keep the impression that blocks disappear from view when these are inserted into holes, a cavity is cut into the polystyrene of the working area. The plate is placed over this hole so that it is fully covered. Still, different positions of the plate are possible since the polystyrene can be moved.
  The blocks can be distributed over the whole working area but must lay on the face that matches a hole. They must not rest on top of each other. This ensures that they can be grabbed and inserted by a movement of the gripper that is perpendicular to the working area (i.e. along the z-axes of the robot).



(a) (b)

Figure 6.3.: Close-up of the robot's end-effector with the two-finger gripper and the attached camera. Panel (b) shows a selection of puzzle blocks and the plate with the matching holes.

- The end-effector is marked with an uniquely colored round patch which is always visible in the static camera. This patch is tracked by the object recognition method in order to estimate an image position of the robot.
- For the experiments described in this thesis the puzzle blocks are wrapped in white paper so that the original color is only visible at the top side of the block. This is made to ease the object recognition process which is based on a simple color segmentation. Without the paper the sides of the blocks could not be properly segmented. Consequently, the position estimation would have become unstable.

6.3.2. Dynamic of the Demonstrator

With the above given static specification, it can be explained how the robot movements look like when the puzzle is played by the system. Four basic behaviors are necessary to move puzzle blocks into the matching holes in the puzzle plate: objects have to be approached, the gripper has to be aligned to them, blocks have to be grabbed or inserted. These four actions are explained in the next subsections.

6.3.2.1. Approaching

Before a puzzle block can be grabbed or inserted, the end-effector must be positioned over it or the matching hole. The necessary basic behavior is a movement of the end-effector in a plane parallel to the working area about 15 cm elevated above it. In the following this plane is called approaching plane. The gripper always faces downwards onto the working area. This movement can starts at any position on the approaching plane and will end at a position where the target object is visible in the end-effector camera.

To solve this basic behavior only the information from the static camera are useful. The target object has to be recognized in the static camera image. It must not be hidden by another object.[2] To realize the movement only the x-y-coordinate of the robot has to be controlled. All other coordinates should be kept constant. Note that the task can not be solved by simply driving the robot so that its image position is equal to the image position of the target object. Since the objects lay on the working field while the end-effector is moving above it, an offset between these two image positions must be preserved. This offset varies with the position of the object due to effects of perspective.

6.3.2.2. Aligning

The goal of the align movement is to drive the gripper directly over a target object and to align the fingers to the shape of it. This basic behavior can only be performed when the target object is visible in the end-effector camera since the pose of the object can only be determined with this camera. So, a precondition of the align movement is the performance of the approach behavior towards the same target object. For the aligning, only the x- and y-coordinate and the roll-angle of the robot have to be controlled. The roll-angle rotates the gripper around the z-axis of the robot. This rotation is needed since the inner sides of the two fingers have to be parallel to one side of the a block or hole in order to grab or insert something into it, respectively. That is the reason why the align behavior has to be performed before a grabbing or inserting movement is commanded.

6.3.2.3. Grabbing

For the grabbing movement the gripper is first vertically lowered towards the working area, then the fingers are closed and the gripper is lifted up again. This movement starts and ends on the approaching plane. At the beginning of the movement the fingers are supposed to be opened and afterwards are holding a puzzle block. The descending is performed in a stepwise manner so that a lateral or rotational displacement of the gripper w.r.t. the object can be corrected. The lowering is controlled by the z-coordinate of the robot, while the displacement correction are performed by changing the x-y-coordinates and the roll-angle.

6.3.2.4. Inserting

The last basic behavior that has to be defined looks very similar to the grabbing movement. The gripper is lowered stepwise from the approaching plane to the working area and back upwards. The only – and obviously needed – difference is that the movement starts with a puzzle block between the two fingers and ends with an opened gripper.

[2] Since the static camera is positioned at the side of the work space it can happen that blocks are occluding other objects.

6.3.2.5. Comments

This quite straightforward and limited way of how a shape-sorter puzzle should be solved with a robot is chosen because of different reasons. First of all, the available two-finger gripper does not allow much more complicated grabbing movements. E.g. it would have been much more effort to cope with blocks that may lay on the wrong side. This would have required that in a first approach the object is picked up and somehow turns onto the right side and than re-grabbed to prepare the inserting movement. To realize this, a more powerful than the available method for object pose estimation would have been needed. Generally, the image processing task was the limiting factor for designing the robot movements. Without appropriate visual information the robot can not be controlled in a sophisticated way. Another reason is that the research interest was laid onto the development of new generic learning methods rather than onto new robot control scheme.

The division of the realized robot movements into the proposed four basic behaviors is coordinated with the tasks of the SP module. The partners concerned with symbolic learning agreed on this problem partitioning so that they could fulfill their research goals. Otherwise, different basic behaviors could have been defined.

6.4. Summary

In this chapter the COSPAL project with its goals and participating partners is outlined. Researchers from four European universities were engaged to develop new learning strategies for a robot vision system. The challenge was to combine continuous and symbolic data processing in one system architecture. New learning strategies were sought that allow the system to be adaptable to its environment throughout its entire run-time.

The developed architecture is presented that contains three main modules: the perception-action, the grounding-management and the symbolic processing module. Basically, the first module is responsible to learn robustly basic behaviors. The second one has to translate continuous into symbolic information (and vice versa). And the third module keeps track of the long-term goals that are learned from human user input. According to these different parts of the system, various forms of knowledge representation and learning scheme were employed (e.g. annotated relational graphs, channel representation, support vector machines, or the HLAM approach).

To test and prove all these abilities, a shape-sorter puzzle was chosen to be solved by the COSPAL robot vision system. The specifications of the realized demonstrator are delineated. The given descriptions of four basic behaviors (i.e. approach, align, grab, and insert) define the problems that are solved with the implementations explained in the next chapter.

Chapter 7

HALMs for COSPAL

In this chapter it is described how the HLAM approach is utilized in the implementation of parts of the COSPAL system. An overview about the tasks that are solved with different HLAMs is given in the first section along with explanations how these HLAMs are embedded in the overall software architecture of the system. In Section 7.2 and 7.3 two submodules are described that accomplish tasks of different complexity in order to realize different robot movements. Finally, an HLAM is proposed that is able to decide which movement is necessary at which point of time to play the COSPAL shape sorter puzzle.

7.1. Overview

In this section subtasks of the COSPAL system are defined that are solved with the HLAM approach. The implemented HLAMs are embedded in submodules within the perception-action module. How these submodules are connected to other parts of the COSPAL software architecture is explained in Subsection 7.1.2. The section is concluded with an overview of the symbols used to explain what information is processed by the various HLAMs.

7.1.1. Objectives

With the implementation of the basic behaviors for the COSPAL system (cf. Subsection 6.3.2) the usefulness of the HLAM approach should be proved. The goal is to show that various aspects of the new approach are beneficial to design networks that can control a robot w.r.t. to visual feedback. Especially the potential of the features that are uncommon to generic machine learning techniques (i.e. hierarchical structure, splitting of domain and model space, and automatic outlier detection) should be examined. To do so, a number of networks are trained to realize the approach, align, grab and insert behavior. Their task is to compute sequences of robot commands (i.e. target positions in the robot coordinate system). As input they use continuous information such as x-y image coordinates of recognized objects or discrete color class labels. Visual information of this kind as well as high-level commands that define what behavior should be performed next stem from outside of the HLAM networks. Such information are given

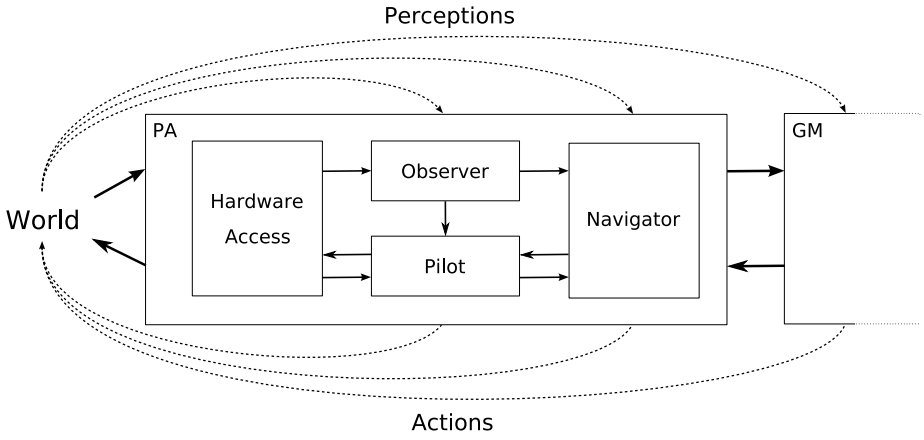


Figure 7.1.: Internal structure of the perception-action module of the COSPAL system. The arrows inside of the PA module indicate which submodules send and receive status information. The arrows outside of the boxes should symbolize that the system realizes perception-action cycles that are working with different scopes of knowledge running with different frequencies.

via certain interfaces that connect the networks with other modules within the COSPAL system architecture. What is relevant to know about these interfaces is explained in the next subsection.

Besides the networks to realize the four basic behaviors, the HLAM approach is used to learn the correct sequence of the puzzle actions. The goal is to train a network that can solve the puzzle by invoking the basic behaviors in the right order. The network should generalize from some examples where the human user was playing the puzzle to its basic idea that first, blocks have to be grabbed and then ought to be inserted into holes with matching shapes. With different premises and different means the problem is solved by the symbol processing module of the full COSPAL system. Despite this fact, the extra task was undertaken with the HLAM approach to prove its applicability to purely discrete data and to exemplify how a trained network can be interpreted.

7.1.2. Internal Structure of the Perception-Action Module

As above mentioned, the HLAM networks for controlling the robot are integrated in the PA module developed at the CAU. They are distributed over two submodules called the *pilot* and the *navigator*. Furthermore the PA module contains the so-called *observer* and *hardware access* submodule. Figure 7.1 gives an overview of the internal structure of the PA module and indicates how these four submodules can exchange information between each other. In the following the tasks of these submodules and their interfaces are explained as detailed as necessary to understand the HLAMs for robot control.

The hardware access submodule is a straightforward interface to the robot and the

two cameras attached to the system. It provides images of different resolutions[1], the position of the robot (x-y-z position and yaw-pitch-roll angles) and status information about the gripper (either closed, totally opened or holding something). On the other hand, it receives target positions for the robot (6D) and commands for the gripper (open or close). For security reasons these positions are checked before executed so that the robot can not be driven outside of the workspace. The hardware access submodule is designed to facilitate the implementation of perception-action cycles in a stepwise manner (i.e. where long range movements are divided into shorter steps in order to visually check their execution). After receiving a command the hardware access submodule blocks the rest of the system as long as the robot has not reached its target position. Hence, the synchronization of concurrent processes that compute robot target positions and access the cameras is not needed.

The image processing routines of the observer submodule solve object recognition and categorization problems. The observer provides different information about recognized objects (i.e. blocks or holes). From the static camera image the x-y position and the color class of the object is extracted. While from the image of the end-effector camera a 3D pose (x-y coordinate and an angle) and three labels are estimated. These labels encode the object's color, shape and type. As color and shape class labels the observer outputs unique integers. Color labels can be red, green, yellow, blue, silver, and black (for the holes). Whereas shape labels are given for cylinders, bridges, triangles, cubes, and crescents. The type label defines ten different classes for all objects of these five shapes: A unique type label is given to each hole and to all blocks with different colors but equal shape. This type label allows a simple discrimination between holes and blocks and the different shape classes.

The observer submodule is supposed to guarantee two conditions. The x-y coordinates of an object in both images have to be constant under rotation of the object. A fixed offset between the computed image coordinates and the actual 3D object is tolerable. On the other hand, the angle extracted from the end-effector camera image can refer to any suitable physical property of the object in any coordinate system (e.g. angle between longest edge and one image axis) as long as it remains consistent over a full turn of the object or the end-effector camera.

From the view point of the robot control submodules it remains important to know that – as a matter of fact – these image information about an object are only available depending on its visibility in the two cameras. Since the end-effector camera depicts only a small portion of the working area, most often, only the object description from the static camera can be given. The case where the object is invisible for the static camera but recognized in the end-effector camera image can only happen when one block occludes another object.[2] This happens rarely since the position of the static camera is sufficiently elevated above the working area and the height of the blocks is relatively low. To acquire all possible visual information about an object the robot has to change something in the scene. In the first case the end-effector camera has to be moved over

[1] (1024×768 pixels for the end-effector and 800×600 pixels for the static camera)

[2] Since holes are positioned on the working plane level they can not occlude other objects.

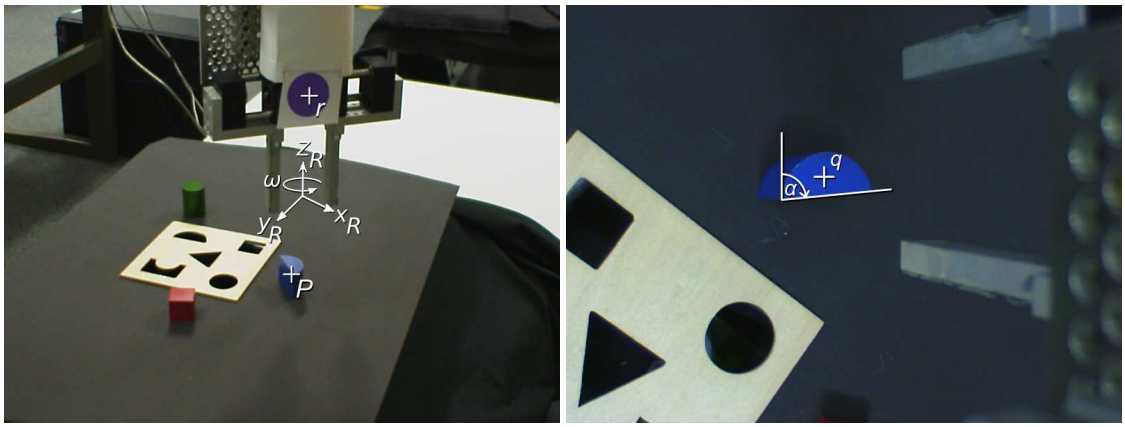


Figure 7.2.: Sample images of the static and the end-effector camera. The image information computed by the observer submodule and given in the variables r, p, q and α are shown.

the object of interest. In the other, the block that occludes another has to be grabbed.

The navigator submodule offers the interface to invoke the four basic behaviors. As mediator for the higher levels of the COSPAL system, the grounding-management (GM) module can initiate the robot movements in order to play the puzzle. All behaviors are performed w.r.t. a puzzle block or a hole. Accordingly, when a behavior is requested by the GM module it has to specify the target object by a system internal identifier that is generated for each object recognized by the observer submodule. In return, the navigator outputs a success or failure feedback after a behavior was performed. In the case something went wrong (e.g. during an insertion move the hole was missed) the navigator ensures that the robot remains in a safe position so that new actions can be started without any danger.

The pilot differs from the navigator submodule w.r.t. to its scope of knowledge about a currently performed behavior. The general idea is that the pilot has a local view onto the task, while the navigator has more extensive responsibilities. This is for example reflected in the fact that the navigator defines the target position of the end-effector while the pilot computes the single steps to reach this position. Significant for this division is that the perception-action cycles realized by the pilot are performed with a higher frequency than those of the navigator. In fact, the navigator encloses the alternations between an image processing step of the observer and a robot movement of the pilot (cf. Figure 7.1). With this principle the pilot realizes for the navigator subtasks of the different behaviors. What and when information is exchanged by these two submodules is explained along with the HLAMs that realize the basic behaviors in Section 7.2 and 7.3.

7.1.3. Nomenclature

The following symbols are used to describe the input are of the HLAMs designed for the pilot and navigator submodule. Figure 7.2 illustrates some of these image information.

Symbols and their meaning:

$r = (x, y)$	Robot position in static camera image
$p = (x, y)$	Object position in static camera image
$q = (x, y)$	Object position in end-effector camera image
α	Object angle in end-effector camera image
$I = (r, p, q, \alpha)$	All image information
$r^*, p^*, q^*, \alpha^*, I^*$	Target position for r,p,q,α and I
$d = (u, v, w)$	Description of object with type, color and shape label
$R = (x_R, y_R, z_R, \theta, \phi, \omega)$	Position of the end-effector in the coordinate system of the robot (the rotation is specified in Euler angles)
G	Status of the gripper
S=(R,G)	Robot state i.e. all robotic status information
B	Basic behavior that is being performed
A	Flag for indicating that aligning is needed (cf. Subsection 7.3.3)

7.2. HLAMs for the Pilot Submodule

The pilot submodule is responsible for subtasks of the approaching and aligning basic behaviors. The navigator invokes the pilot whenever the robot has to be to steered to certain target positions. How the HLAM approach is utilized to implement these tasks is described in the next subsections.

7.2.1. Generic Visual Servoing Scheme with Online Refinement of Transfer Function

The pilot realizes a visual servoing scheme[3] that is able to drive the robot to a target position that is given in image coordinates. The basic idea is to implement an iterative process that gradually steers the robot towards the target position under continually visual control. According to the taxonomy proposed in [112], the realized process is an *image-based, static look-and-move* visual servoing scheme. It is image based – and not position based – because the information that is used to compute an error signal to control the robot is given in the image coordinate system. In contrast to that, a position based system would first transform the information into a world coordinate system before a robot command could be determined. The implemented visual servoing scheme complies to the static look-and-move paradigm because it depends on a robot controller that is

[3] The term visual servoing stems from [54]. It refers to control schemes that use image information as error signals to control a robot. For an overview see [73].

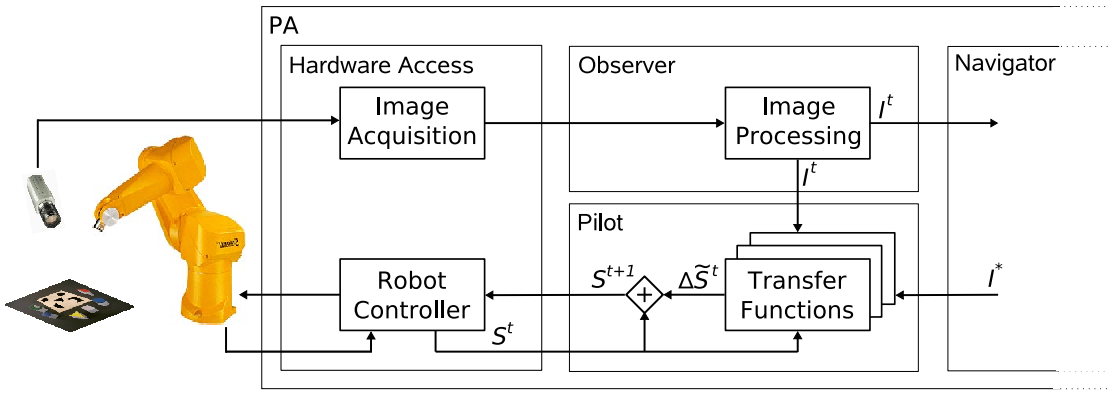


Figure 7.3.: The control paths of the image based, static look-and-move visual servoing scheme as implemented with the pilot submodule.

independent from the visual processing. The controller receives a target position in its own coordinate system and steers the robot to it by means of an autonomous control loop.

In Figure 7.3 the realized visual servoing scheme is depicted. Note how the control loop for the robot is enclosed by the image based control loop. The figure shows in details those parts of the PA module that are important for the pilot submodule. It makes obvious that the base of the pilot is defined with different transfer functions. In every cycle t of the visual servoing loop, these functions map the target image position I^*, the most up-to-date image information I^t and robot state S^t to a robot command ΔS^t. This command defines how the robot has to change its position in order to reach or at least get closer to the target position. It is executed by the robot controller that is a part of the used RX90 system.

So, the key to realize the needed movements to play the shape sorter puzzle are the above mentioned transfer functions. The functional structure stems from generic control theory. These transfer functions are specifically trained HLAMs. For the pilot submodule three different HLAMs are realized: one to let the robot approach an object, and two others to align the gripper to it. As it will be described below, these HLAMs use different image information (e.g. only positions from the static camera) and control different parameters of the robot (e.g. only the roll angle). But they are applied in the same generic algorithm that implements the visual servoing scheme and an online refinement of the transfer function. Its pseudo code is given in Algorithm 7.8.

Depending on the needed movement (for an approach or align behavior) the algorithm is invoked by the navigator submodule with an image target position and one of the three transfer functions.[4] It realizes the above described visual servoing loop that essentially

[4]To be most general, these transfer functions are defined with I^*, I^t and S^t as input arguments. But the actual implemented transfer functions use only some components of these information.

Algorithm 7.1: Pseudo code of the online learning visual servoing control scheme that steeres the robot to the target image position I^*. With a specific control function f a specific movement can be realized.

Function `ServoRobotToTargetPosition`
Input : I^*, f
begin
  $t \leftarrow 1$
  $I^t \leftarrow$`GetCurrentImageInformation()`
  while $(\|I^t - I^*\| > \epsilon) \ \wedge \ (t < T)$ **do**
    $S^t \leftarrow$`GetCurrentRobotAndGripperStatus()`
    $\Delta S^t \leftarrow f^t(I^*, I^t, S^t)$
    $\Delta \tilde{S}^t \leftarrow$`EnsureValidity(`$\Delta S^t$`)`
    `DriveRobotTo(`$S^t + \Delta \tilde{S}^t$`)`
    $I^{t+1} \leftarrow$`GetCurrentImageInformation()`
    $f^{t+1} \leftarrow$`UpdateTransferFunction(`$f^t, I^t, I^{t+1}, S^t, \Delta \tilde{S}^t$`)`
    $t \leftarrow t + 1$
  if $\|I^t - I^*\| < \epsilon$ **then**
    return *success*
  else
    return *failure*
end

alternates status acquisition with robot movements. This loop runs as long as the sensed image position differs from the target position by a certain amount or too many attempts were made to reach it. For both criteria the individual thresholds (ϵ and T) have to be chosen appropriate for the application. To secure the system the robot command ΔS^t computed by the transfer function is processed by the procedure `EnsureValidity`. This procedure outputs the finally executed command $\Delta \tilde{S}^t$. It can be applied to limit the magnitude of the movement or to check that all robot parameters are within their valid range.

Besides driving the robot to a target position the algorithm enables the system to learn from its actions. In each cycle the information how the world looked like before and after the robot moved can be utilized to refine the transfer function. This is reasonable because of the following principles. The transfer function essentially maps a difference in image space to a difference in the robot coordinate system. The difference in image space is the misplacement between a sensed image position I^t and the given target position I^*. While the difference in the robot coordinate system – the executed command $\Delta \tilde{S}$ – is representing the movement of the robot that is needed to compensate the misplacement in image space. On the other hand, every movement of the robot produces a change in the image position. It delivers a sample how a difference in the robot coordinate system is related to a difference in image space. In short, a movement reveals the correspondence between an executed robot command and the resulting change in image space and exactly this relation has to be established by the transfer function. Hence, each movement can

be employed to refine the transfer function. Such a correction is always appropriate when changes of the environment are possible or the function was only coarsely established in the first place. As discussed below, both reasons apply for the COSPAL system.

Algorithm 7.8 is defined to steer the robot along a straight trajectory to a static position. But it can easily be extended to realize movements where a dynamical target is tracked or curved trajectories are pursued. Only the target image position has to be altered to be a function of time (i.e. $I^*(t)$). By this means an obstacle avoidance behavior could also be realized. Like in [85], the desired trajectory could be modeled with a virtual force vector field which is a superposition of an attractor and a number of repellor vector fields. The attractor would define the actual target, while the repellors prevent the collision with detected obstacles. Nevertheless, such dynamic movements are not realized with the COSPAL system since – pragmatically – it was not necessary for the puzzle and – more important – it was no scientific goal of this thesis to improve the state of the art of dynamic tracking systems.

7.2.2. Transfer Function for Approaching

In order to realize the approaching behavior the navigator invokes the servoing algorithm of the pilot. At such a point of time the end-effector is positioned on the approaching plane (i.e. it is elevated to a fixed height above the working area). The navigator specifies the target position r^* of the approach movement in coordinates of the static camera image. It assumes that the pilot steers the end-effector to that position without leaving the approaching plane. To do so, only the x-y components of the robot state S have to be changed (cf. Section 6.3.1). The pilot realizes this movement with a HLAM which input and output are specified in Table 7.1. The definition of the input relies on the possibility of the HLAM approach to employ different domain and model spaces. The domain space is the coordinate system of the static camera so that different local models are responsible for different local regions of the static camera image. These models map the difference between the most up-to-date image position r^t of the robot and its target image position r^* to a robot state S. This state defines how the robot changes its position to get at least closer to the target position. The training of this HLAM guarantees that all but the x-y-components of S are zero so that the end-effector remains positioned on the approaching plane. The applied safety procedure `EnsureValidity` checks that the norm of S does not exceed a certain threshold. By this means the step size of a robot movement in one cycle is limited to be smaller than e.g. 2 cm. This results in a movement that is homogeneously interrupted for possible corrections to reach the target position.

This HLAM is trained during a bootstrapping and the above described online refinement phase (i.e. whenever the navigator invokes the servoing algorithm for an approaching movement). The network is bootstrapped with training samples that are acquired with a small number of pre-specified movements. To do so, the end-effector is driven to some training positions on the approaching plane where its position r in the static camera image and the state S are saved (see Section 8.3 for exact number). This results in a set $P_{\text{Approach}} = \{(r_i, S_i)\}$ of correspondences of robot positions in the image and the robot coordinate system. With two different samples i and j of P one training

Table 7.1.: Specification of the HLAMs designed for the pilot submodule. The first two rows show what the networks receive as input, while the last row defines the output.

	Approach	Align Rotation	Align Translation
$x^{(M)}$	$r^t - r^*$	$\alpha^t - \alpha^*$	$q^t - q^*$
$x^{(D)}$	r^t	$(q^t, z_R^t)^T$	$(q^t, z_R^t)^T$
$f\left((x^{(M)}, x^{(D)})^T\right)$	ΔS^t	ΔS^t	ΔS^t

Table 7.2.: Definitions for the three HLAMs of the pilot submodule how a training sample $\left((x^{(M)}, x^{(D)})^T, y\right)$ is generated out of two training positions i and j (cf. Subsection 7.2.2 for more explanations).

	Approach	Align Rotation	Align Translation
$x^{(M)}$	$r_i - r_j$	$\alpha^i - \alpha^*$	$q^i - q^*$
$x^{(D)}$	r_i	$(q^i, z_R^i)^T$	$(q^i, z_R^i)^T$
y	$S_j - S_i$	$S_j - S_i$	$S_j - S_i$

sample $\left((x^{(M)}, x^{(D)})^T, y\right)$ for the HLAM is created according to Table 7.2. In this way a training set for the HLAM is produced with all pairwise combinations of samples from P. Given this set the HLAM can be bootstrapped with one of the learning algorithm presented in Chapter 3. During the refinement phase, with each cycle of the control loop one training sample is generated in a way corresponding to the definitions of Table 7.2. With this sample the network is updated by means of the HLAM online learning algorithm.

The HLAM basically learns the correspondence between the difference of two image positions and the difference of the same positions in robot state space. Mainly due to perspective effects these correspondences vary at different positions of the robot. Two robot steps of same size will result in different position changes in the camera image depending on the distance of the end-effector to the camera. Similarly, lens distortions will change the appearance of robot movements between different local regions of the camera image. Because of this dependencies the domain space of the trained HLAM is the image space of the static camera. It allows that the local correspondences between position changes in the image and robot coordinate system are reproduced with single models. Since the HLAM approach allows to specify a different domain and model space this knowledge about the learning problem can be introduced in a straightforward manner into its solution.

Another benefit of the proposed visual servoing scheme is its easy extensibility to movements with more degrees of freedom. E.g. movements in 3D can simply be realized

by adding a second static camera to the system[5] and driving the robot to more positions during the bootstrapping phase. The additional camera is needed to resolve ambiguities between 2D image and 3D robot positions. The extra training positions have to reflect that also the z-component of the robot state can vary. Everything else of the servoing algorithm and training method can remain unchanged.

When comparing this learning based servoing approach with the literature, it becomes obvious that the HLAM approximates the pseudo inverse of the so-called image Jacobian [37, 60, 73]. This matrix expresses the differential relationship between the camera and the robot coordinate frame. It is normally derived from a certain camera model (e.g. pin hole camera with radial lens distortion). Such assumptions imply that the external and internal camera parameters have to be calibrated whenever something in the experimental setup changes. With the above proposed learning scheme the bootstrapping phase replaces the first of such calibrations. Any other needed recalibration can be omitted as the servoing algorithm ensures that the system adapts itself to environmental changes. The quality of this visual servoing scheme is validated in Section 8.3.

7.2.3. Transfer Functions for Aligning

The task of the pilot during an aligning movement is to drive the end-effector so that a given object is visible in the end-effector camera image at a certain x-y position with a certain angle α. Like for the approaching movement the navigator specifies these target positions, this time with q^* and α^*. The translational part of the movement should be performed in planes that are parallel to the working area. These planes can have different distances to the working area but reach maximally the height of the approaching plane. This type of movement can be achieved with varying the x-y-components of the robot state. The needed rotation can be realized with an adjustment of the roll angle since the gripper is always pointed downwards.

For the translational and the rotational part of the aligning movement, two different HLAMs are employed as transfer functions. Both are trained (i.e. bootstrapped and afterwards refined) according to the same principle as the HLAM for the approaching movement. They also approximate image Jacobian matrices. But this time the differential relationship between the coordinate system of robot and the end-effector camera – not the static camera – is needed. To realize this, different input spaces for the HLAMs are defined and (partly) different robot state components are changed for the training positions. See Table 7.1 for the specifications of both HLAMs.

Again, the domain spaces differ from the model spaces. Both HLAMs use with their domain space the image position q^t and the z-component of the robot state S^t to allow a specialization of single models of the HLAM to local regions of the end-effector camera image. The argumentation is the same as for the approaching movement: effects of perspective and distortions due to lens irregularities have to be compensated by the local models. To successfully cope with perspective it is important to include the z-component of the robot state into the domain space. Again, the phenomenon is that the same robot

[5] With a second static camera, the image coordinates r and r^* will be extended by a second x-y coordinate pair.

step results in a different displacement of the object in the image coordinate system. But unlike the approaching movement this difference does not only depend on the absolute image position but also on the distance of the camera to the object. Exactly this distance is expressed with the z-component of the robot state.

Depending on the task the input of the two HLAMs are different. The one for the rotational part of the aligning movement takes the difference between the sensed angle α^t and the target angle α^* as input. While the input for the translational part is the difference of the sensed and the target position q^t and q^*, respectively. The procedure `EnsureValidity` for the rotation checks that the roll angle ω stays in a secure range. This is necessary because the experimental setup does not allow a full turn of the end-effector. At one point the end-effector camera would crash into the robot arm. A consequence of this limitation is that a commanded aligning movement will fail if the object is badly positioned. The `EnsureValidity` procedure for the translational HLAM fulfills the same goals as the one for the approaching movement by restricting the calculated robot steps to a maximal size.

These HLAMs are bootstrapped with the same strategy as for the approaching movement but with different kinds of training position. For the HLAM responsible for rotation a set of samples $P_{\text{Rotation}} = \{(q_i, S_i)\}$ has to be acquired where only the roll-component ω of the robot states S is changed. Given such a set, the training sample for the HLAM are generated as defined in Table 7.2. For the other HLAM the end-effector is driven to the corners of a cube that is parallel to the working area. By this means a set $P_{\text{Translation}} = \{(q_i, S_i)\}$ is produced where the x-, y- and z-components of the robot state vary as needed for the aligning movement. Again, the training samples for the HLAM are generated as specified in Table 7.2. But this time it is important that only those samples i and j are used for a new training sample that have the same value of the z-component in the robot state. Otherwise the resulting transfer function would steer the end-effector not just in the x-y plane but also along the z-axis of the robot.

Note the similarity of the definitions and training between these three HLAMs for the approaching and aligning movement. This makes it obvious that various types of movements can be realized with the same principle. Only quite simple customization for driving to different training positions and for employing different inputs in the HLAMs are needed.

7.3. HLAMs for the Navigator Submodule

Within the COSPAL architecture the navigator submodule has the responsibility to realize the four basic behaviors. The symbol processing (SP) module invokes via the grounding-management (GM) module the navigator whenever an approach, align, grab or insert action should be performed. The SP module defines when and w.r.t. which visible object a basic behavior is needed to play the puzzle. The navigator offers an interface to the GM module to invoke the procedures that realize the actions by means of three additional HLAMs and the servoing algorithms of the pilot. Via this interface the navigator receives one identifier for the desired behavior and one identifier that references

the object that should be e.g. grabbed. The former is saved in the variable B that is needed in one HLAM of the navigator. While the latter is used to set up the variables p, q, α and d with the image information about the reference object computed by the observer submodule. The navigator sends a success or failure feedback to the GM module. The possible errors of the different behaviors are described along with their implementation further below.

The navigator employs three different HLAMs to achieve the above described tasks. One network computes the target position of the end-effector so that the pilot can let the robot approach the reference object. Similarly, another network defines the target position in the end-effector camera needed for an alignment of the gripper to the object. Last but not least, one HLAM is trained to steer the gripper vertically in order to grab an object or to insert one into a hole. These networks and their training schemes are described in the next three subsections.

7.3.1. HLAM for Approaching

The approaching behavior is invoked whenever an object that should consecutively be e.g. grabbed is only visible in the static camera image. The robot is supposed to be driven in the approaching plane so that the object is recognized in the end-effector camera image. The steering is realized with the pilot submodule as described in Subsection 7.2.2. What is needed to start the visual servoing algorithm is the target position r^* for the end-effector in the static camera coordinate system. Obviously, this target position depends on the position of the reference object. As noticeable in the left panel of Figure 7.2, the position of the object p and the target position r^* differ from each other by a certain offset. This offset is depending on the position of the object on the working area and on the type of object. The former is a result of the perspective: the offset must be larger when the object is closer to the camera. The later is due to the weak assumptions how the observer is supposed to compute x-y positions of different objects. E.g. for the same position on the working area holes have typically a smaller y-component as blocks because holes are flat and blocks have a certain height. These variations of the offset have to be modeled in order to compute the target position of an approaching behavior.

It is done with an HLAM that makes use of the possibility to structure an HLAM hierarchically. The network is depicted in Figure 7.4. The subnetworks in the lower level learn the object specific dependency of its position p and the target position r^*. While the second level is used to switch between different objects according to their classified color label[6] v. In this second level the mechanism for automatic outlier detection (see. Subsection 3.3.5) is applied for reasons described further below. The output of the hierarchy is computed with:

$$r^* = \hat{f}\left((p,v)^T\right) = \sum_{l}^{L} g_{\mathrm{EG}}\left(a_{2,l}(v)\right) \sum_{k} g_{\mathrm{EG}}\left(a_{1,k}(p)\right) \hat{y}_{1,k}(p), \tag{7.1}$$

[6] The color label is the only information about the type of an object available from the static camera image (cf. Subsection 7.1.2).

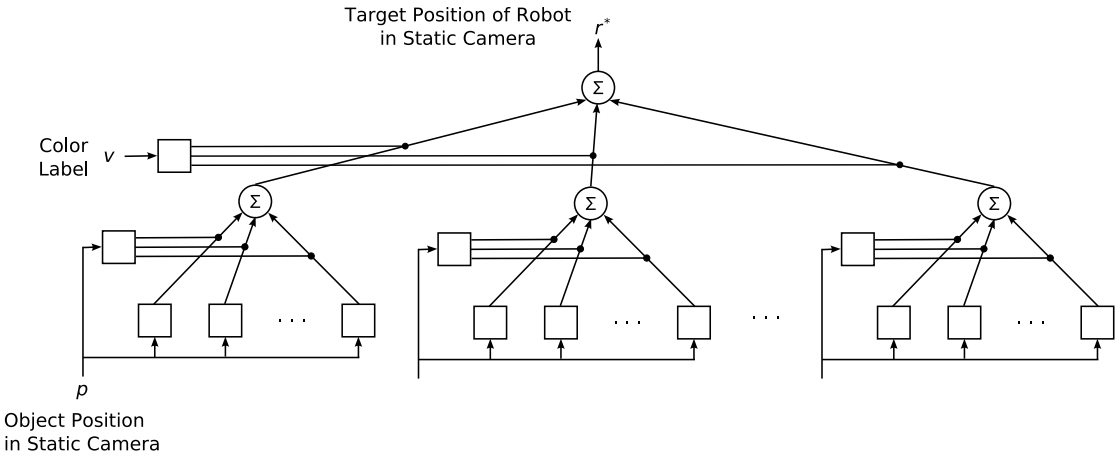


Figure 7.4.: Two level HLAM that computes the target position of the robot for an approaching movement. The subnetworks on the lower level are specialized to different objects.

where L is the number of trained objects, $g_{\mathrm{EG}}(\cdot)$ is the exclusive gating law, $\hat{y}_{1,k}(\cdot)$ and $a_{1,k}(\cdot)$ are the local models and their activations functions of the first level and $a_{2,l}(\cdot)$ are the activations functions of the second level. The number of needed local models in the lower level is not given since it varies within the object type.

This hierarchy is established by two means. The local models and the activation functions of the first level are trained individually with the regular learning algorithms of the HLAM approach. To do so, for each object a single set of samples $\{(p, r^*)_i\}$ is acquired. As described in Subsection 3.1.4, the gained networks are afterwards combined by defining the activation functions $a_{2,l}(\cdot)$ with one of the domain parameter estimation algorithms proposed in Section 3.3. The color class labels of the previously collected sample sets are used as input for such an algorithm.

The acquisition of the samples sets is done during a bootstrapping and an online refinement phase. With the first initialization of the system the end-effector is driven to a few positions on the approaching plane where the human user had to place objects so that these are recognized in the end-effector camera. By this means the first samples (p, r^*) along with the color class label of the recognized object can be collected for the above mentioned training sets. These sets are used to create the first version of the HLAM to realize approaching behaviors. It allows movement towards those objects that are contained in the first training sets.

During run-time the HLAM can be refined and extended to new objects whenever one is recognized in the end-effector camera. During an approaching movement, with each step commanded by the pilot the observer checks if an object is visible in the end-effector camera. If this is the case a new sample (p, r^*) is acquired and used to update the two-level HLAM. To do so, the ability of the so far trained HLAM is tested by using p as input to estimate an output. The automatic outlier detection mechanism will throw an exception if – up to this point of run-time – no subnetwork has been trained for the

recognized object. In such a case a new subnetwork is created for this yet unseen type of object. Otherwise the subnetwork with the highest activation value $a_{2,l}(\cdot)$ (i.e. the network that corresponds to the color class label of the object) is updated with the new sample by means of the online learning algorithm for HLAMs.

This network for computing the target position of the robot for an approaching movement utilizes important features of the HLAM approach. It exemplifies how a hierarchy of subnetworks can be created. It shows how different information sources can be employed at different levels of the hierarchy. Its first level comprises HLAMs that are individually trained. These are combined by activation functions on the second level that are set up with the regular domain establishing algorithm. During run-time, the automatic outlier detection mechanism serves useful to decide when a new subnetwork should be added to the hierarchy or only a single subnetwork should be refined. An added subnetwork extends the competences of the HLAM without interfering with already established abilities.

Furthermore the automatic outlier detection mechanism is needed to produce a failure feedback for the GM module. An error signal will be sent by the navigator whenever it was commanded to approach an object that has not been recognized in the end-effector camera during run-time of the system. Such a case is detected by the automatic mechanism applied to the activation functions of the second level. Another error the navigator can produce stems from the pilot submodule when too many visual servoing steps were performed to approach the object (cf. Subsection 7.2.1). In all other cases the navigator submodule will report success to the GM module after the pilot has reached its target position.

7.3.2. HLAM for Aligning

An alignment of the end-effector – and with it the gripper – to a reference object is needed to prepare a grabbing or inserting behavior. As explained in Subsection 6.3.2, during a grabbing or inserting behavior the robot is only controlled vertically (i.e. along the robot's z-axis). This makes it necessary that during the aligning behavior the end-effector has be correctly positioned in the x-y plane (i.e. on a horizontal plane) and the roll angle is appropriate for the object. The navigator realizes these tasks in a very similar manner as the approaching behavior. It first computes by means of an HLAM the target position for the reference object and then invokes the pilot to steer the robot to it. This time the HLAM has to estimate the target angle α^* and the target position p^* given in the coordinate system of the end-effector camera image. A failure feedback is supposed to be produced when the reference object is unknown to the navigator or not visible in the end-effector camera, or the pilot needed too many visual servoing steps.

The aligning behavior is invoked by the GM module after a successful approaching behavior was performed. In such a case the end-effector is positioned on the approaching plane. But correct aligning is supposed to work also at different heights of the end-effector over the working plane (for reasons see the next subsection). This makes it necessary to employ a machine learning technique instead of a look-up table that could store for each object one target position and angle. The input and output specifications of the HLAM

Table 7.3.: Specification of the HLAMs designed for the navigator submodule. The first two rows show what the networks receive as input, while the last row defines their output.

	Targeting for Aligning	Grab/Insert
$x^{(M)}$	z_R	z_R
$x^{(D)}$	u	$(u, G, B)^T$
$f\left((x^{(M)}, x^{(D)})^T\right)$	$(q^*, \alpha^*)^T$	$(\Delta S, A)^T$

that solves this are given in Table 7.3.

The input $x^{(D)}$ for the activation functions is the type label u. It allows that the local models are specialized to different objects. The local models map from the z-component of the robot state to the needed target information α^* and p^*. This separation of the model and the domain space is chosen because it is obvious that the target positions change continuously with the height of the end-effector to the reference object and that the different objects can have quite different target positions.[7] The type label u is chosen to group local models in the domain space because of two reasons. On one hand, it allows to separate all objects according to their shapes in a way that also holes and blocks with the same shape are discriminated. And on the other hand, it groups blocks with the same shape but different color. That reduces the number of needed local models to a minimum.

Like in the case of the HLAM discussed in the last subsection, the automatic outlier detection mechanism is the base for one error signal. Whenever the type label u does not match an established domain, a failure is sent to the calling module. An error is also sent when the reference object could not been recognized in the end-effector camera image or the pilot stopped with an error.

The HLAM for aligning is trained with samples of the form $\left((u, z_R)^T, (q^*, \alpha^*)^T\right)$ which are acquired during a procedure described in the next section that is also used for the HLAM needed for the grabbing and inserting behavior. The basic idea is to position the objects directly underneath the gripper, drive the robot to different heights and to store after each step the required image and robot state information. In a bootstrapping phase the offline learning algorithm for HLAMs is employed to create the network with a set of samples for different objects. During run-time the network can be extended to other types of objects without interfering with the already established competences. Given another set of samples, new local models and corresponding domains can again be established with the offline learning algorithm and simply be added to the network. Since the type label is a unique integer the network will work properly like before the extension. This demonstrates that the HLAM approach is useful to realize a kind of semi look-up table where MLTs are generically organized by labels.

[7] Especially the target angle varies significantly over the different objects due to their unique shapes.

7.3.3. HLAM for Grabbing and Inserting

The grabbing or inserting behavior is commanded after an approaching and aligning action w.r.t. to a certain reference object was successfully performed. As described in Subsection 6.3.2 the movements to grab and to insert a block are very similar. For both, the gripper is lowered from the approaching plane to the working area and after a closure or opening of the two fingers the gripper is lifted up again onto the approaching plane. Lowering and lifting of the gripper is a vertical movement where only the z-component of the robot state is varied. The lowering is divided into steps where horizontal movements or rotations around the z-axis can take place to correct the pose of the gripper w.r.t. the reference object. For both behaviors the navigator submodule is supposed to report a failure in four different cases: either the reference object is not visible in the end-effector camera, the navigator has not been trained for the type of object, no aligning behavior has been performed to it, or at one lowering step a misplacement could not be corrected by the pilot. A grabbing can also fail if the block slips out of the gripper's fingers. Whereas in case of an insertion an error signal is sent when the gripper is not holding anything after the fingers were closed.

For these two behaviors the navigator submodule employs one additional HLAM. The key to realize both behaviors with one mechanism is to rely on the following strategy to insert blocks into holes. The implementation assumed that all blocks with the same shape are always grabbed in precisely the same manner. E.g. for grabbing a cube the fingers of the gripper are always positioned in the middle of it and not sometimes closer to a corner. With this assumption it can be omitted to check the pose of a grabbed object while performing an inserting behavior. Hence, grabbing and inserting can be performed with the same HLAM since only one reference object – either a block or a hole – has to be taken into account (i.e. the HLAM can have the same input space). The specifications of the realized HLAM is given in Table 7.3.

This HLAM is employed in a modified version of Algorithm 7.8. Again, the HLAM computes in a loop the next robot command ΔS^t. The network is trained so that only the z_R and the gripper state component G of ΔS^t can have non-zero values (i.e. the robot is lowered or lifted and the gripper can be closed or opened). In contrast to the servoing scheme of the pilot submodule, the servoing loop runs as long as $\|\Delta S^t\|$ is larger than a certain threshold and not as long as $\|I^t - I^*\|$ is larger than ϵ. This stopping criterion is implemented because the end-position of a grabbing or inserting behavior can not be determined with only visual information as simple as for the functionalities of the pilot. The HLAM for grabbing and inserting also computes a flag A that defines whether a horizontal or rotational misplacement should be corrected after the robot was moved a step along its z-axis. According to this flag, the servoing algorithm invokes the aligning behavior described in the last subsection.

The inputs of the HLAM are the z-component of the robot state, the state G of the gripper, the type label u of the reference object, and the flag B that defines whether a grabbing or inserting behavior is being performed. Only the robot's z-component is used as input of the local models, the remaining data defines their domains. This separation between the model and domain space is suited to learn classes of movement

that are specific for the type of object and whether a grabbing or inserting movement is commanded. Consequently, each class can be realized with a single local model.

Training samples for this HLAM are acquired in a procedure where the human user teaches the system how to either grab or insert an object. To do so, the end-effector must be positioned on the approaching plane directly over either a block or a hole of the shape sorter puzzle box. To train to grab a certain type of block the gripper must be aligned to a sample block so that the end-effector can be lowered without any horizontal or rotational correction in order to grasp it. Correspondingly, the gripper must be manually aligned to a hole before an insertion movement can be learned. After this preparation the user can interactively define a sequence of steps where the end-effector is lowered or lifted and the gripper is closed or opened. He or she can define how many such steps should be performed in order to grab or insert a certain block and if or if not an aligning movement is needed at a step. One training sample for the HLAM specified above can be generated at each lowering or lifting step. To do so, the user has to supply some information. In the variable B the type of behavior that should be trained is stored as an integer. The user defines how long and in which direction (up or down) the desired movement along the robot's z-axis should be in the next step. This information is used for the ΔR part of the target output value ΔS. The other part (i.e. ΔG) is defined by the decision of the user to open, to close or to leave the state of the gripper unchanged. Last but not least, he or she defines with a target value for A if an aligning movement should be performed or not. If a misplacement should be corrected one training sample for the aligning HLAM is acquired. As described in the last subsection the current image information about the object (i.e. position and angle in the end-effector camera) is taken as a sample for a correctly positioned object.

With a number of sequences for different objects a set of such training samples can be acquired during a bootstrapping phase. The gained set is used with the offline HLAM learning algorithm to train the HLAM for the grabbing and inserting behavior. During run-time of the system the established network can be extended to cope with other types of objects. To do so, new training sequences have to be performed and again, the offline learning algorithm is used to establish new local models for the collected training sample. These local models and their domains can simply be added to the already working HLAM. This does not corrupt its performance on objects trained at an earlier stage. Similar to the HLAM described in the last subsection, this competence extension works nicely on the situated time scale since the object type label u employed to define the domain for a local model allows a clear distinction between different object types.

7.4. Learning the Puzzle Game from Human Demonstration

As an extra task within the COSPAL scenario the HLAM approach is used to learn to play the puzzle from examples how a human user sorts the blocks into the matching holes. This task is performed by a module that invokes the functionalities of the perception-action module (i.e the four basic behaviors realized by the navigator and the object recognition methods offered by the observer). The module has to ensure that the right basic behaviors

are performed in the correct sequence and w.r.t. appropriate reference objects. It replaces the GM and SP module of other COSPAL partners so that a standalone system could be realized that combines the two work packages of the CAU: the PA module and the implemented functionalities to recognize human movements (cf. Subsection 6.2.3).

In the next subsection it is described how the human movement recognizer is supposed to generate training data that expresses the game play of the puzzle. It follows the specification of two HLAMs that can learn the correct sequence of basic behaviors. This section is concluded with the explanations how these HLAMs are used to play the puzzle fully automatically.

7.4.1. Acquiring Samples of the Puzzle Play from Visual Demonstration

The key to an efficient learning scheme is a high-level representation of the correct game play. When many variations of the real-world situation (e.g. different positions of the blocks) can be expressed with a single code (i.e. symbol) less training samples are needed to learn the similarities between game situations. The set of the four basic behaviors and the different object type, color and shape labels represent already such an abstract view onto the game. So, what is needed is a method to acquire sequences of – in the following called – command samples. Such a command sample defines which behavior was performed w.r.t. what kind of reference object. For example a red cylinder would be correctly grabbed and inserted into the black round hole by the command sequence: `approach red round; align red round; grab red round; approach black round; align black round; insert black round`. Such a sequence exemplifies how one block is properly handled. A number of sequences is needed to reveal that there is a common pattern within these behavior invocations and object attributes. This pattern is specific to the game. The methods outlined in Subsection 6.2.4 can be used to produce such sequences by observing a human user who is demonstrating the game. These methods were still under development when this thesis was written. Hence in the following only the plan can be explained how the methods should be connected to the actually implemented HLAMs described in the next subsection.

Within an extra module these methods can be trained to recognize human movements that correspond to the four basic behaviors. While a user is handling different blocks in order to sort them into the holes the movement recognizer module would divide the trajectory of the user's hand into segments. By matching such segments to the basic behaviors one can acquire a sequence of labels for the observed actions. At the same time the characteristics (i.e. the color and shape) of the reference object of each action can be determined. To do so, the area of the image where one movement segment ended could be analyzed with the methods of the observer submodule: whatever object could be recognized in the vicinity of the user's hand should be taken as the reference object. So, by compiling sequences of both, classified trajectory segments and color and shape labels of the recognized reference objects, the correct game play can be encoded in a high-level representation.

7.4.2. HLAMs to Learn the Correct Sequence of Basic Behaviors

A number of HLAMs are designed to play the shape sorter game. More specifically, the HLAMs are used to generate a sequence of commands sent to the PA module so that a certain block is grabbed and inserted into the right hole.[8] The whole puzzle can be solved by repeating such a sequence for all blocks that are in reach of the robot. Which block is handled at which point of time is finally irrelevant but must be specified when such a grabbing-inserting sequence should be performed. To do so, the user has the possibility to enter the color and shape label the next to be handled object should have. By this means he or she can supervise the system. Still, such supervising information have not necessarily be produced by a human user. The system itself can decide which block should be grabbed and inserted next. How this works is described in Subsection 7.4.3.

After defining what information is available (i.e. sequences of command samples acquired by visual demonstration and the supervising information) the strategy to learn the puzzle game can be explained. The basic idea is to learn with the HLAM approach a transfer function that maps a game situation to a command that is appropriate to be executed. A game situation summarizes the history and the presence of the system. While the computed command represents the future of the system. For the shape sorter puzzle, a game situation is defined with the last executed command, the type, color and shape attributes of the last reference object and the supervising information. Given such a game situation the transfer function has to output which basic behavior should be executed and which color and shape attribute the next reference object should have. By repeatedly applying this transfer function the system will play the puzzle. Each execution of a command will change the game situation so that the correct sequence of actions will be reproduced.

For the shape sorter puzzle it is sufficient that the history of actions that is encoded in the game situation description includes only the last action. This will be shown in experiments described in Section 8.5. For more complicated tasks the description of the game situation could easily be extended to former commands and more information about the present state of the system.

The transfer function is realized with two different types of HLAMs. In a preprocessing step, one type is necessary to transform the available information about a game situation into a representation that serves as input for the second HLAM type. The latter computes the final output of the needed transfer function. What these types of networks exactly realize is explained in the following two subsections. How they work together is described in Subsection 7.4.3.

7.4.2.1. HLAM for Transforming the Information about a Game Situation

For each basic behavior one HLAM of the type needed for the preprocessing step has to be trained. Such an HLAM determines whether the basic behavior the network represents is applicable to a certain object or not. E.g. the HLAM trained for the grabbing behavior would indicate that a red and round object can be grasped, while that is not the case for

[8] Correct grabbing and inserting includes the approaching and aligning movements.

a black and round object. These HLAMs essentially learn the relationship between the visual information about objects and the possibility to perform actions with them. To do so, the input of such an HLAM contains the type, the color and the shape label of an object.[9] The output is a binary flag that denotes the applicability of the basic behavior.

Training samples for this type of HLAM can be generated with sample sequences that are acquired with the human movement recognizer module. These sequences are given as a set of samples where each sample contains a label for the performed basic behavior and the visual attributes of reference object. This set of samples has to be ordered w.r.t. the type, color and shape label of the reference objects. This results in subsets of samples that have the same object description but different action class labels. For each of these subsets one training sample can be generated for each preprocessing HLAM. As input of such a training sample the objects description unique in the subset has to be taken. The output is either positive or negative depending on whether the subset contains or does not contain a sample with an action class label that corresponds to the basic behavior specific HLAMs.

For further processing the outputs of the different HLAMs for a certain object are combined in a vector. This vector will have the same values for all types of blocks but different ones for holes. Just because, in contrast to holes, blocks can be the reference object for the approaching, aligning and grabbing but not for the inserting behavior. Such a vector is an object description that allows to group (i.e. to classify) objects w.r.t. their utility in the shape sorter puzzle. This is helpful to learn the transfer function as described in the next subsection.

7.4.2.2. HLAM for Learning the Shape Sorter Transfer Function

One HLAM is trained to realize the needed transfer function to let the robot play the puzzle. As already mentioned the network's input comprises the supervising information (i.e. color and shape label of the desired reference object) and the last game situation. Latter is the label of the last performed basic behavior and the output of the preprocessing HLAM computed for the visual description of the last reference object. Only the supervising information serves as input for the local models of the HLAM. Whereas the models' domains are established with the description of the last game situation. The output of the HLAM is the basic behavior next to be performed and the color and shape specification of the reference object.

Again, the sample sequences from the visually demonstrated game plays are the base to produce training samples for this HLAM. In contrast to the training of the preprocessing HLAM, the order of the command samples is important. Two successive command samples are used to generate one training sample for the transfer function HLAM. The first sample delivers the needed information about the last game situation. Whereas the second sample defines the target output since it represents what has to be done after the system has reached a certain situation. For all samples that belong to one grab-insert sequence (i.e. a sequence that starts with an approaching of a block and ends with an

[9] The model and the domain space coincide.

inserting behavior) the same supervising information is taken. It is the color and shape label of the block that was grabbed during the sequence.

Furthermore, an extra training sample is generated for each grab-insert sequence. Its input is the game situation about the finally performed inserting behavior and the supervising information that describes the inserted block. The target output that defines the next command is a code that represents a stop action. It explicitly marks the end of a grab-insert sequence. The realized shape sorter puzzle player explained in the next subsection uses this stop action to divide a course of game where a number of blocks are inserted. This division allows that a user can let the system handle single blocks he or she wants to get inserted. Without this stop action the number of commands for one sequence must have been hard-wired by the system designer. Furthermore the number of commands would have to be constant for all sequences, whereas the explicitly marked end of a sequence enables the system to learn different types of command sequences. This is not needed in the actually solved shape sorted puzzle but could be very helpful. E.g. a special type of block could require an extra behavior that rotates the block before it can be inserted properly. Without changing the code for the system, this additional behavior can be inserted into the command sequence and it will be learned by the transfer function HLAM.

7.4.3. Automatic Shape Sorter Puzzle Player

As already outlined above, the system can grab and insert a block into the matching hole by a repeated invocation of the trained transfer function. To do so, a loop is programmed that has a similar structure as the visual servoing algorithm applied in the pilot submodule: at the beginning of each cycle, the situation the system is in is determined, given this information the next command is computed with the transfer function, the command is executed, and then the loop can start again. In the following the loop is explained in more details.

The information about the situation in which the system is has to comprise the last performed action, the description of the last reference object and the supervising information. The first two parts are simply memorized from the last cycle of the loop. The supervising information is determined once before the loop starts as it remains constantly valid throughout the full grab-insert sequence. Three options are implemented how the color and the shape label of the designated reference object is determined. Either the human user selects both of them from a list of available colors and shapes. Or he or she defines only one of these two categories so that the system decides about the missing one. Or finally, the system fully automatically selects both, color and shape features. The first option is implemented in a straightforward manner with a simple console where the user types in the wanted color and shape specifications. If he or she leaves one feature category unspecified, the system completes the supervising information in accordance to the blocks distributed on the working area. To do so, the observer submodule generates a list of all visible blocks, this list is compared with the shape or color label specified by the user, and one block with the correct feature is selected randomly to supply the missing label. By this means it is possible that the user can only type in `red` and all

blocks that are red but have an arbitrary shape are sorted into the matching hole. The system works in a similar manner when it should play the puzzle fully automatically. For one grab-insert sequence it chooses a pair of shape and color labels from one block of the list of available objects.

The command next to be executed is computed by the HLAMs explained in the last subsection. First, the HLAMs for preprocessing are used to transform the information about the game situation (i.e. the last used reference object) into the binary vector that describes which behavior can be performed for this type of object. This vector, the memorized last action and the supervising information serves as input for the transfer function HLAM. The HLAM's output is the next command which is then executed. To do so, the computed shape and color label of the reference object is matched with the list of objects recognized by the observer. This is done in order to determine the system internal identifier that specifies the reference object when the navigator submodule performs the commanded basic behavior.

Such a sequence of assessing the game situation, computing the next command, and executing it is repeated in a loop as long as the output of the transfer function HLAM is not the stop action. After this loop came to an end (i.e. after a block was grabbed and inserted properly), the supervising information can be changed and the loop can be started again. The result will be that all blocks are cleared off the working area and the puzzle is solved.

7.5. Summary

In this chapter it is described how the HLAM approach is applied in the COSPAL system. The implementation of the four basic behaviors (approach, align, grab and insert) and a module that coordinates these behaviors in order to play the shape sorter puzzle is explained. The realized HLAMs are distributed in two submodules – called pilot and navigator – of the perception-action module within the COSPAL system architecture. The interplay between these two and the other modules is described in order to show how the robot can be controlled with interleaving perception-action cycles.

With the pilot submodule an image-based, static look-and-move visual servoing algorithm is realized that is able to steer the robot to a target position that is given in image coordinates. This algorithm is the generic base for robot movements that are needed for the approaching and aligning behavior. Specific movements are accomplished with different transfer functions that are specially trained HLAMs. These networks are refined during the operation of the system by means of the online learning algorithm. This ensures that the commonly required thorough calibration of the visual robot system can be omitted. Instead a coarsely bootstrapped system can start operation and will refine itself during run-time to changes of the environment. The definitions of the applied HLAMs make use of the possibility to separate the domain and the model space. This helps to specialize the local models of a HLAM to a local region of the camera images so that effects of perspective and lens distortion can be compensated. As shown with three different type of movements, the proposed training scheme and servoing algorithm offers

a simple principle to learn specific robot control schemes.

In the navigator submodule, HLAMs compute target positions where the robot should be driven to by the pilot. The implementations utilize that information of different degrees of abstraction can be distributed in a hierarchy of locally arranged models. Typically the subnetworks of the realized HLAMs are trained for specific types of objects. This makes it possible that during run-time new subnetworks can simply be added to the hierarchy. This extension on the situated time scale to new types of objects does not interfere with already established competences. Furthermore, the automatic outlier detection mechanism is used to trigger such competence extensions and to generate failure signals in cases where the HLAM is not suitably trained to handle a certain object.

Finally, a mechanism grounded on different HLAMs is proposed that lets the system play the shape sorter puzzle fully automatically. Its base is an HLAM that serves as a transfer function that maps from the history of the puzzle play and symbolic supervising information to the next executed basic behavior. It demonstrates that the HLAM approach is also suited for purely discrete information sources as the training set is a sequence of labels of actions and object specifications. To solve the puzzle play task a new description scheme for the objects is proposed that is needed for a preprocessing step. With it different HLAMs learn in a generic way to transform the image information about an object to information that are action related.

Chapter 8

Experiments with the COSPAL System

The HLAMs proposed for the COSPAL system are validated in different tests. Most of these are qualitative tests due to reasons explained in the first section of this chapter. The results of these general experiments with the system are reported in Section 8.2. Two more detailed validations are conducted for parts of the pilot and the navigator submodule (cf. Section 8.3 and 8.4). For the last section the HLAMs for the automatic shape sorter puzzle player are tested. The open question was to which extend the networks can generalize from grab-insert sequences for example blocks to arbitrarily colored and shaped blocks. The proof for successful learning is given with the thorough analysis of the realized HLAMs.

8.1. Setup and Goals of the Experiments

In the beginning of this section some additional comments are made about implementation details of the COSPAL system. These are necessary since they ground some physical restrictions of the experiments. Relying on these, the goals of the conducted experiments can be explained in Subsection 8.1.2.

8.1.1. Modified COSPAL System

The in the following described experiments were performed with a slightly modified version of the COSPAL demonstrator (cf. Section 6.3). The physical setup had to be changed due to limitations of different software modules. At the point of time when the experiments for this chapter were conducted only a preliminary version of the observer submodule was available. Its object recognition methods are based on color segmentation of the HSV images of the two cameras. It is supposed that each object is colored uniquely. This makes it possible to determine the needed information about an object from the set of connected pixels of the object's color. The center of mass of such a uniformly colored image region gives the x-y-coordinates of the object for both the static and the end-effector camera. The angle α of the object corresponds to its longest edge that is determined with the Hough transform on the data of the segmented end-effector camera image. The shape and type label of the object is not extracted from the images but defined in a look-up table that is indexed with the recognized color label.

To get robust results with this image processing methods, the holes with the half-cylinder and the bridge shape are customized for the experiments. These holes are marked with colored patches next to the actual cavity in the puzzle plate. The attained values for the x-y-coordinates and the angle α are varying for fixed positions of objects about 1 pixel and 0.75°, respectively. This results in an uncertainty of about 1-2 mm and 1° of the robot's position which is appropriate to handle the used blocks and holes with the robot (cf. Section 8.3).

8.1.2. Goals of Experiments

For this thesis only the functionalities of the COSPAL system that are realized with the HLAM approach are of concern. These are mainly implemented with the navigator and pilot submodule of the perception-action module so that most of the tests were performed to validate the robot control schemes. Those tests were conducted on the real robot in order to ensure the functional reliability of the different submodules. The primary goal was to prove that the HLAMs proposed in the last chapter are able to realize the four basic behaviors. For the whole COSPAL project it was more important that these abilities are supplied by the PA module than that these are working in the most efficient manner. Precision was only demanded in qualitative terms of successful insertions. It was no scientific goal to realize a control scheme that is able to insert with the minimal number of servoing steps blocks into perfectly matching holes that would allow only a minute displacement of the blocks. Besides this general objectives highly precise robot control schemes would have not been achievable with the available image processing methods (cf. last subsection).

According to these general goals, most of the tests with the system are of a qualitative nature as summarized in the next section. Another reason why thorough experiments about speed and precision were omitted is that the COSPAL system with its unique setup is practically not comparable to other systems described in the literature. Nevertheless two selected functionalities of the PA module are validated in quantitative experiments (see Section 8.3 and 8.4). They exemplify with detailed numbers how the costs of training of the HLAMs are related to the gained performance. This should demonstrate that the proposed learning based robot control schemes are competitive to classical approaches.

Furthermore the HLAMs of the in Subsection 7.4.3 proposed automatic shape sorter puzzle player are validated on a simulated COSPAL system. The experiment is not performed with the real robot system for three reasons. First, the training set had to be synthesized since the modules for human movement recognition (cf. Subsection 7.4.1) were not fully available when this thesis was written. Second, because of the modular architecture of the system, the functionality of the HLAMs trained to play the puzzle can be verified without the physical execution of the computed commands. Last but not least, the simulation is only a simplification of experimental effort but not of the complexity of the task since the HLAMs work on purely discrete data that can perfectly be synthesized and validated. Besides the prove of functionality the experiment should highlight how a trained HLAM can be fruitfully interpreted.

8.2. Qualitative Tests

The functional reliability of the PA module is proven with numerous sequences where a block is first grabbed and then inserted into the matching hole. To ease this testing, most often the four basic behaviors needed for this sequences are manually invoked with the interface the PA module offers (cf. Subsection 7.1.2). This avoids semantically wrong commands issued by the SP and GM module. Nevertheless the PA module also demonstrates in different runs its abilities with the fully connected COSPAL system.

For these qualitative tests the two blocks and the puzzle plate with the holes are placed at various positions on the working area. This ensures that the HLAMs needed for the approaching behavior are able to steer the robot to arbitrary positions including those at the border of the working area. For the other behaviors it is not important to test such extreme positions since their execution is not depending on information extracted from the static camera image but rely only on those of the end-effector camera. The only effect that can be observed for different object positions is that the lightening conditions vary over the working area. This causes sometimes problems as the color segmentation becomes unstable. Especially during grabbing behaviors the center of mass (i.e. the x-y-coordinates) of the segmented image region shifts significantly but inconsistently due to the shadows that are casted by the gripper's fingers onto the bock. Nevertheless the precision of the image and robot control methods prove to be sufficient to insert the blocks into their holes.

To ascertain that the proposed HLAMs can be specialized to various objects it is sufficient to train them for the two blocks and two holes. The different training procedures are performed consecutively for the four objects like it would be done for any other block or hole. As described in Section 7.3 the unique type label of the objects ensures that these four and any other are properly handled by the system.

The training of the different HLAMs successes quite easily. The effort for bootstrapping the pilot submodule is small and the online refinement works fast and properly (see Section 8.3). The most time demanding part is the training procedure for the grab and insert functionality as it has to be performed individually for each of the four objects. Still, with a quite limited number of training samples (maximal six per object type) the HLAM can solve the task.

Generally, the configuration of the trained HLAMs (e.g. the choice of a domain model) is easy and uncritical. All HLAMs work with the hyper-elliptical domain model and most often with simple linear local models. Similarly, the selection of the required meta-parameters is easy and can be done intuitively. E.g. it is quite simple to choose a realistic value for the meta-parameter ϵ that defines the maximal RMSE of a local model since the output space of the networks is typically the coordinate system of the images or the robot. In the next sections it is exemplified what effort is needed and what success is achievable for training an HLAM of the navigator submodule.

Table 8.1.: Meta-parameters for HLAMs applied in experiments described in this chapter.

	minSamples	ϵ	*maxBufferSize*
Targeting End-Effector	2	1 [pixel]	×
Approaching	5	2 [pixel]	15
Preprocessing for Puzzle Player	1	0.0001	×
Puzzle Player	1	0.0001	×

8.3. Validation of Visual Servoing Algorithm

The quality of the generic visual servoing algorithm proposed in Subsection 7.2.1 is assessed with the HLAM designed for the approaching behavior. The main question of interest is how successful the online refinement of the transfer function works. Unfortunately the trained HLAM can not easily be validated since its ground truth (i.e. the proper mapping from image to robot space) is unknown. At most one could compare the new approach with classical methods which would involve extensive camera calibration. A different way for quality assessment is chosen since such classical methods were not available and too costly to be implemented. In the following an experiment is described where the end-effector is steered by the pilot submodule to different test positions and the resulting trajectory is judged w.r.t. the number of needed servoing steps and its straightness.

The HLAM specified in Subsection 7.2.2 is bootstrapped with eight training samples. To do so, the end-effector is driven over a 3×3 grid at the center position of the approaching plane. The grid's step size is 5 cm. At each vertex of the grid the position of the end-effector is recorded in the coordinate system of the robot and the static camera image. With this data, eight training samples for the HLAM are generated which correspond to the eight possible movements from the center position of the grid to the border positions.[1] These movements are depicted in Panel (a) of Figure 8.1. These few training sample are used to bootstrap an HLAM configured with hyper-ellipsoids as domains and linear models as local models. The selected meta-parameters for the applied online learning algorithm are listed in Table 8.1. Like for the HLAM described in the last section, these parameters are not optimized since they worked directly from the start.

To prepare the validation of the trained HLAM the robot is driven manually to four test positions. These are situated at the corners of the 26 cm×36 cm approaching plane and serve as target positions of the test trajectory. For each corner the position of the end-effector is recorded in the coordinate system of the robot and the static camera image. The former allow a comparison with the positions actually reached by the robot during testing, while the latter are needed to specify where the pilot should drive to.

Given the image coordinates of the four test positions, the robot can be steered by the

[1] To allow a simpler visualization of the training set, the generation of training samples described in Subsection 7.2.2 is slightly modified. It is more restrictive since more than eight training samples could be generated from nine test positions.

Table 8.2.: Mean number of visual servoing steps needed to reach four test positions.

Trial #	1	2	3	4	5
Bootstrap Camera Position	13.4	10.2	5.8	6.4	5.6
Shifted Camera Position	10.6	7.2	6.8	7.5	5.6

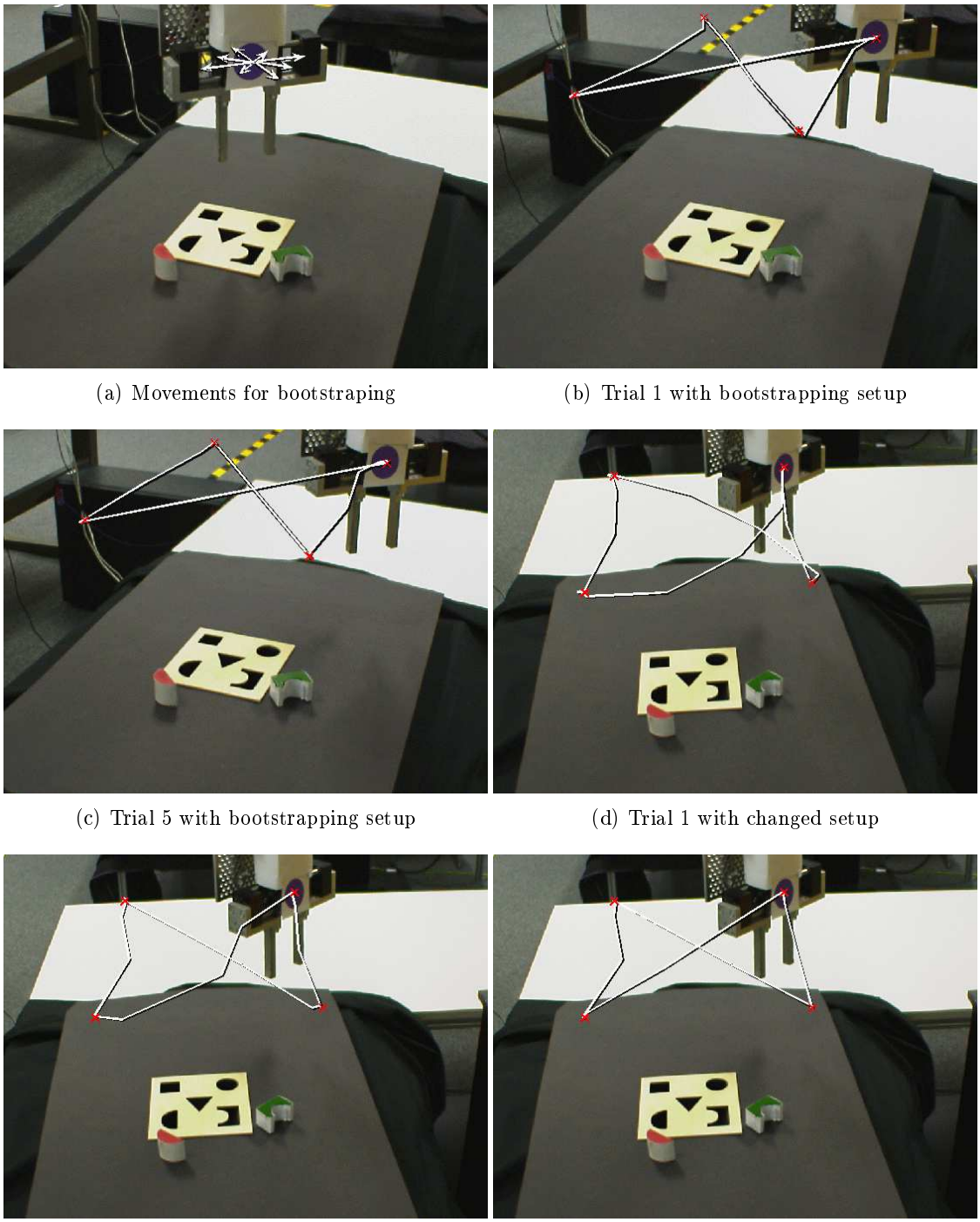
visual servoing algorithm along the test trajectory which is a loop that connects every corner. The thresholds ϵ and T of the servoing algorithm are selected so that the robot is driven to the target position with the highest possible accuracy. To do so, ϵ is set to zero which means that no misplacement of the end-effector from its target position is permitted. The maximal number of servoing steps T is left unrestricted so that as many steps as needed can be performed. The safety procedure `EnsureValidity` limits the step size to 15 cm.

With these settings the robot is driven five times along the test trajectory. In all trials all positions were reached. The trials differed only w.r.t. the number of needed visual servoing steps per target position and the straightness of the trajectory. In Panel (b) of Figure 8.1 it is visible that in the first trial the robot is not steered along the ideal way on straight lines to the target positions. After the fifth trial the realized trajectory plotted in Panel (c) is improved. The effect that the online refinement increases the quality of the HLAM can also be demonstrated with the number of needed servoing steps. In Table 8.2 the mean numbers of steps per target position are listed for each trial. The mean number drops from 13.4 to 5.6. It shows that the HLAM learns to steer the robot faster along the test trajectory.

To test the robustness of the proposed visual servoing algorithm, the static camera is displaced from its original position and five more trials are performed with the bootstrapped HLAM. The static camera is shifted sideways by approximately 40 cm, lifted by 10 cm and tilted by 10°. Again, as preparation for the actual trials the robot is steered manually to the four test positions. These are the same physical positions as in the first five trials. But without their new image coordinates the pilot could not drive the robot to them.

The trajectory of the first trial is plotted in Panel (d) of Figure 8.1. It shows that the robot clearly deviates from the ideal path. Still, all target positions are reached with a speed that is comparable (even slightly better) to the speed with the original camera position (cf. Table 8.2). In Panel (e) and (f) the results of the second and the final trial are plotted. It demonstrates that the performance of the HLAM improves rapidly. With the fifth trial the same quality is realized as for the camera setting that is used to produce the bootstrapping training set. This proves that the HLAM approach is suited for fast online learning.

Since the test positions are also recorded in the coordinate system of the robot, the precision of the system can be quantified. The mean displacement over all ten trials is 2.1 mm. The maximal and minimal error is 2.2 mm and 1 mm, respectively. These errors are totally depending on the object recognition methods since the visual servoing loop does not stop before the target position is accurately reached (due to setting ϵ to zero).



(a) Movements for bootstraping

(b) Trial 1 with bootstrapping setup

(c) Trial 5 with bootstrapping setup

(d) Trial 1 with changed setup

(e) Trial 2 with changed setup

(f) Trial 5 with changed setup

Figure 8.1.: Effects of refining the approaching HLAM of the pilot submodule. Panel (a) shows the static view with the eight training samples for bootstrapping the HLAM. In Panel (b) and (c) the trajectory is plotted how the robot drives to four test positions (red crosses) in the bootstrapping setup. The servoing results after the static camera is shifted and tilted are shown in Panel (d)-(f).

A closer look at the data reveals that the misplacement of the robot is higher at target positions that are more distant from the camera. This matches well the conception that precision is mainly a matter of camera resolution.

8.4. Validation of Estimating the Target Positions for Aligning

One experiment is performed with the HLAM that computes the target positions of objects in the end-effector camera. As described in Subsection 7.3.2 this HLAM has to estimate the target x-y-coordinates p^* and the target angle α^* depending on the type of object and the height of the end-effector over the working area. Samples for learning are normally acquired along with the training procedure for the grab and insert functionality (cf. Subsection 7.3.3). To facilitate a validation of the targeting HLAM this procedure is modified. The different objects are still placed directly underneath the end-effector so that with a vertical movement the gripper is driven to a position appropriate for a grabbing or inserting. But in contrast to the normal training procedure the vertical lowering movement is divided into more steps for the experiment. At each step the position of the object is measured in order to record test and training samples for the HLAM. These allow a quantitative validation of the proposed network.

For the two blocks and two holes, four of such vertical movements are performed. Each is a lowering of the end-effector by 10 cm from the approaching plane down to the working area with a step size of 1 cm. This yields 44 samples of target positions of the object at 11 different heights above the working area. For each object three samples were used as training and the remaining eight as test samples. The training samples belong to the highest, the lowest and the middle position of the end-effector during the vertical movement. Figure 8.2 depicts the samples for the two blocks.

The HLAM is configured with the hyper-elliptical domain model and the exclusive gating law. The model validation criterion for the used offline learning algorithm is based on the training error. In the first test linear models and afterwards polynomials of order two are utilized as local models. The chosen meta-parameters of the learning algorithm are listed in Table 8.1. With the linear models a RMSE of 6.7 pixel is realized on the test set. With the quadratic polynomial the RMSE is reduced to 0.9 pixel (cf. Figure 8.2). The latter proved to be sufficient to realize proper grabbing and inserting.

This functionality of the navigator is realized very conveniently. The finally used configuration of the HLAM is found with only these two tests comparing different local models. The selected domain model and the meta-parameters are the first choice and work perfectly.

8.5. Validation of the Shape Sorter Puzzle Player

The HLAMs proposed to learn the game play of the shape sorter puzzle (cf. Subsection 7.4.3) are tested w.r.t. their ability to handle new types of objects properly. The question is whether the networks can generalize from examples where blocks with some

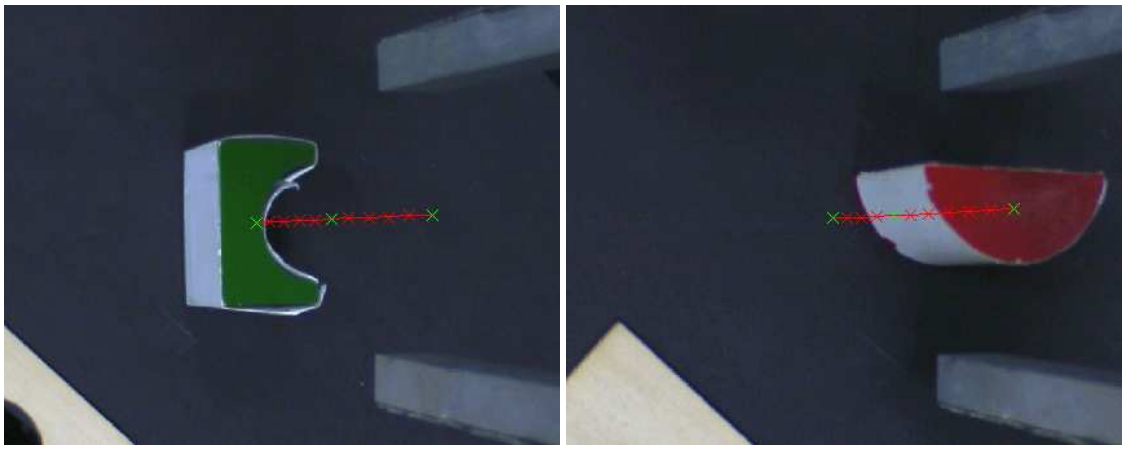


Figure 8.2.: Target positions in the end-effector camera of two blocks. Each cross belongs to a position of the robot at a different height over the working area. In the left image the end-effector is situated on the approaching plane, while on the right it is lowered ready to grab the red half-cylinder. The green crosses show training samples for the HLAM that approximates the function plotted with the two lines. As visible the deviation between test samples (i.e. red crosses) and the approximation is negligible.

colors and shapes were inserted to blocks that can have arbitrary shapes and colors. The subsequent question is how this competence is achieved.

As argued in Subsection 8.1.2 synthetic training and test data is generated for this experiment with the automatic puzzle player. This data is a sequence of commands and supervising information that shows how the puzzle game is properly played with the four basic behaviors realized by the perception-action module. The manually generated sequence agrees perfectly with a sequence that would be acquired with designated method described in Subsection 7.4.1. This is possible since the encoding of the commands and supervising information is purely discrete (i.e. symbolic). Command sequences are generated for all blocks available in the physical world. With five different colors and four different shapes, 140 ordered command samples with corresponding supervising information are synthesized. Only a small portion (i.e. 21) of them are used as training samples. These are listed in Figure 8.3.

The training samples demonstrate how the game is played with three different blocks. They show that for all blocks a fixed sequence of seven commands (i.e. `approach; align; grab; approach; align; insert; stop;`) has to be issued. Furthermore the general pattern for the color and shape attributes of the reference object becomes obvious. E.g. for the first six commands the shape label is equal to the corresponding supervising information. Or the attributes of the reference object for the stop command are always set to the label `none`. This set of samples is selected because it contains enough information to allow a generalization to arbitrarily colored and shaped blocks. It demonstrates for the learning algorithm that different color and shape labels are possible and

```
 1: Approach red   bridge        (supervisor: red  bridge)
 2: Align to red   bridge        (supervisor: red  bridge)
 3: Grab     red   bridge        (supervisor: red  bridge)
 4: Approach black bridge        (supervisor: red  bridge)
 5: Align to black bridge        (supervisor: red  bridge)
 6: Insert   black bridge        (supervisor: red  bridge)
 7: Stop     none  none          (supervisor: red  bridge)
 8: Approach blue  bridge        (supervisor: blue bridge)
 9: Align to blue  bridge        (supervisor: blue bridge)
10: Grab     blue  bridge        (supervisor: blue bridge)
11: Approach black bridge        (supervisor: blue bridge)
12: Align to black bridge        (supervisor: blue bridge)
13: Insert   black bridge        (supervisor: blue bridge)
14: Stop     none  none          (supervisor: blue bridge)
15: Approach blue  half-cylinder (supervisor: blue half-cylinder)
16: Align to blue  half-cylinder (supervisor: blue half-cylinder)
17: Grab     blue  half-cylinder (supervisor: blue half-cylinder)
18: Approach black half-cylinder (supervisor: blue half-cylinder)
19: Align to black half-cylinder (supervisor: blue half-cylinder)
20: Insert   black half-cylinder (supervisor: blue half-cylinder)
21: Stop     none  none          (supervisor: blue half-cylinder)
```

Figure 8.3.: Training set for the puzzle player. For the HLAMs the command, color and shape labels are represented as an integer but translated to their meaning in this figure.

that these are specifically related to the supervising information.

With this training set and the meta-parameters selected as given in Table 8.1, five HLAMs for the preprocessing step and the HLAM for the transfer function are trained (cf. Subsection 7.4.2). All networks are configured with the center domain model and the exclusive gating law. Their local models are linear models. The five HLAMs for the preprocessing step correspond to the five possible commands: `approach, align to`, `grab`, `insert`, and `stop`. The latter is added to the four basic behaviors to explicitly mark the end of a grab-insert sequence. The corresponding HLAM is trained by the same means as those for the basic behaviors. It learns to separate any real world object – block or hole – from a pseudo object which color, shape and type label is `none` because this labeling occurs only together with the stop command.

The remaining 119 command samples are used to test the performance of the trained networks. They show a 100 % correct reproduction of command sequences to grab and insert blocks with any combination of color and shape. This ensures that the automatic shape sorter puzzle player will invoke the right basic behaviors for whatever block the human user selects the supervising information.

Analysis of the HLAMs: To understand how this solution works the trained HLAMs are examined in details. As already mentioned, the five networks for the preprocessing step learn to distinguish three different classes of objects by their color, shape and type label. They separate holes, blocks and the pseudo object from each other. This is reflected in the fact that they produce together exactly three different output vectors: either $(1,1,1,0,0)^T$, $(1,1,0,1,0)^T$ or $(0,0,0,0,1)^T$ where each component of the vector corresponds to the output of one HLAM which are ordered as "approach", "align to", "grab", "insert", and "stop". The vector $(1,1,1,0,0)^T$ defines the class of blocks since the commands `approach`, `align to` and `grab` can be invoked for them but not the commands `insert` and `stop`. To this vector the output of the HLAMs for holes varies only in the third and fourth component as holes can not be grabbed but inserted. Quite different to that is the responds for the pseudo object: the vector $(0,0,0,0,1)^T$ shows that only the `stop` command can be issued for an object which color, shape and type label is `none`.

Keeping these three classes of objects in mind, the HLAM trained as transfer function can easily be interpreted. The network contains seven local models. Their position in the domain space of the HLAM is defined with the parameter p of the center domain model. This vector has six dimensions and reflects the information about the last game situation (cf. Subsection 7.4.2). One dimension corresponds to the last performed action, while the other five take as input the output of the preprocessing HLAMs. Since it is clear that the second part defines three different classes the position of the seven local models can be plotted into a 2D diagram as shown in Figure 8.4. One dimension corresponds to the invoked behavior. The other represents a projection of the 5D output space of the preprocessing HLAMs onto a 1D subspace that discriminates the blocks, holes and the pseudo object. In Figure 8.4 this dimension is labeled with "Color, Shape" as these two labels are essentially the inputs to the preprocessing HLAMs. According to this, the block class is labeled with "any but black, any", the holes with "black, any" because

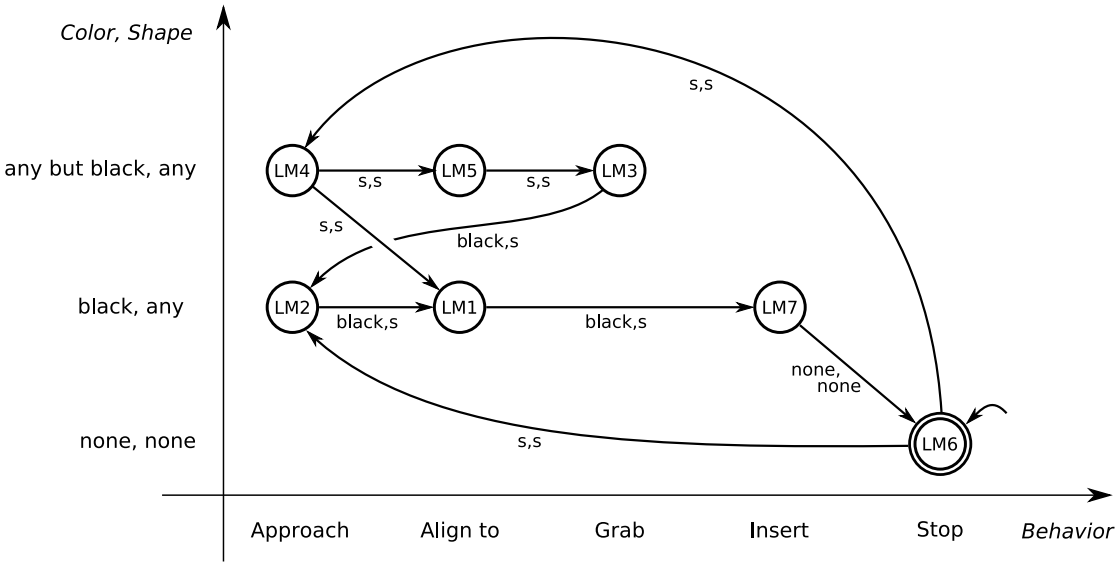


Figure 8.4.: Finite state automaton learned by the transfer function HLAM. Each circle corresponds to a local model. The vertical axis reflects three classes of objects. The edge annotation specifies the color and shape attributes of the reference object where "s" means that the supervising information is taken. The local model LM6 is the start and end state.

the only difference between the two classes is that holes are black. The pseudo object is labeled with "none, none".

Taking the parameter β of the linear models into account, this diagram can be extended with arrows that represent the output of the local models. A simple example is the local model labeled LM7 that is selected to compute the output of the HLAM whenever the last performed behavior was an insertion and the last reference object was a hole. This local model has a fix output defining that the stop action has to be performed next and that the shape and color label of the new reference object are both `none`. Hence, an arrow with the annotation "none, none" is drawn from LM7 to LM6 since the position of the later in the domain space represents the command "`stop none, none`". A more complicated example is the local model LM4 which computes an output when in the last situation a block was approached. The parameter β of LM4 defines that the next action is an aligning behavior and that the color and shape labels of the next reference object are set to the values of the supervising information. Correspondingly, two arrows start at LM4. One points at LM5 which represents aligning movements for blocks. The other arrow points at LM1 which stands also for the aligning behavior but this time w.r.t. an hole.

So, the whole diagram can be interpreted as a finite state automaton. The drawn local models represents states, while the arrows define the state transitions. This automaton is realized by the loop described in Subsection 7.4.3 that repeatedly applies the trained HLAM to compute the next command. In which state the automaton is in at a certain

point of time is decided with the gating law. It selects this local model to compute the output of the HLAM which position p matches to the input of the activation function (i.e. the last situation description). As exemplified above the next command (i.e. the output of the local model) is determined by the parameter β of the selected local model. This command is executed and the next cycle of the loop will start. Since the just performed command defines the new input of the HLAM the automaton will traverse through its possible states until the stop command has to be executed and the loop ends. As the local model LM6 corresponds to this stop command it is marked as the finite state of the automaton.

Additionally, LM6 represents the start state of the automaton for a correct grab-insert sequence. When the loop starts, the last commanded behavior is set to the stop action and the labels of the last reference object to `none`. This is conform with the training set where the approach commands issued for new grab-insert sequences (cf. line 8 and 15 in Figure 8.3) are preceded by stop commands.

With this explanations the working principle and the correctness of the automatic shape sorter puzzle player becomes obvious. A full grab-insert sequence is performed along the path from LM6 through LM4, LM5, LM3, LM2, LM1, LM7, and finally back to LM6 again. With the first three commands a block is approached, the gripper is aligned to it and it is grasped. The attributes of the reference object for these commands are defined by the supervising information until the second approach command is computed by LM3. At this stage of the puzzle play the color label is selected to be `black`, while the shape label remains to be defined by the supervising information. By transversing from LM3 over LM2 and LM1 to LM7 the puzzle player invokes the approaching, aligning and inserting behavior for a hole with the same shape attribute as the grabbed block. With reaching LM6 again, the stop action is computed and the grab-insert sequence is finally completed.

The diagram contains two arrows that are unnecessary to play the puzzle. In fact, the edge between LM6 and LM2 and the edge between LM4 and LM1 could lead to wrong commands. They would allow that the approaching and aligning behavior is invoked too early for a black object. Instead of grabbing a block, the system would start with trying to insert something into a hole. These extra edges exists because of two reasons. First, the HLAM contains two local models for the approaching and the aligning behavior. Second, the parameter β of the local models LM6 and LM4 defines that the reference object has the same labels as given by the supervising information. So, if the supervisor misleadingly selects the color label to be black, the finite state automaton will produce a wrong command sequence. That this is not happening is ensured in the COSPAL system by the procedure that generates the supervising information.

8.6. Summary

The experiments described in this chapter demonstrate that the HLAM approach is successfully applied in the COSPAL system. In accordance to the general research goals of the COSPAL project and due to some restrictions of software modules that do not

depend on HLAMs, the success of the proposed methods is asserted with qualitative tests. Such tests show that the implemented perception-action module is able to grab and insert blocks properly. They also exemplify that the realized HLAMs can easily be configured. In all cases the appropriate domain models, the type of used local models and the meta-parameters of the learning algorithms are selected without extensive optimization procedures.

The visual servoing algorithm implemented with the pilot submodule is validated with the HLAM needed for the approaching behavior. It shows that its online refinement scheme works fast and robustly. Even strong shifts of the static camera can be compensated with a small number of training steps. Equally good is the performance of the navigator submodule to compute target positions for the aligning behavior. The responsible HLAM generates outputs with sub-pixel accuracy.

Finally, the proposed shape sorter puzzle player demonstrates powerful generalization ability. It produces for any proper supervising information the correct sequence of commands to grab and insert the corresponding block. This is proven with an analysis of the trained HLAM where the network is interpreted as a finite state automaton. To do so, the positions of the local models in the domain space are regarded as the states, while their output defines the transitions between the different states. This is simplified by the HLAM features that the model and domain space can be separated and the exclusive gating law offers a simple model (i.e. state) selection mechanism.

Chapter 9

Conclusions

Finally, the content of this thesis is summarized and concluded with an output of possible future work.

9.1. Summary

Part I: The "Hierarchical Network of Locally Arranged Models" (HLAM) – a new supervised machine learning technique (MLT) – is proposed and validated. Its name indicates its basic idea that locality is the key to built successfully solutions for learning problems. Locality refers to the principle that the individual models of the network contribute to the solution only in their so-called domains which are local regions of the input space. Their approximation of the target function is only valid in rather small portions of the input space. Global validity is ensured by local specialization.

The advantage of locality is simplicity. Since an individual model has to be valid only in its domain it can be realized with a simple technique which performs the needed mapping from the input to the output space. In turn a simple mapping technique eases the analysis of a built solution. Knowing the region a single model is responsible for and knowing how the model works, one can understand how the whole network accomplishes its task. The possible hierarchical structure of an HLAM also facilitates such an analysis. Local models can be grouped together so that they define a larger but still connected region of the input space. This can be done at different levels of a hierarchy. This allows a grouping of models w.r.t. different degrees of similarity.

To realize these ideas it is most important to answer two basic questions: How can a domain be defined so that it is clarified which model of the network is responsible for what data? And, by which means is the input space split up into such local regions? One would wish that a domain can arbitrarily be shaped and still its parametric model is computationally easy to establish. On the other hand, the ultimate goal for the process of splitting up the input space is to find regions that are as large as possible but can still be governed by single local models. By fulfilling these objectives one can approximate target functions efficiently since the input space can be divided w.r.t. the complexity of the target function. More local models will approximate the target function in smaller regions where the function is complicated. Whereas less models are needed for larger regions where the target function can be approximated with less effort.

In Chapter 2 of this thesis machine learning techniques are reviewed that adhere to the locality principle. These MLTs commonly define domains as continuous weighting functions that are used to mix the output of the local models. The mixing weights are functions of the input of the target function. This makes it possible that the models contribute in different regions of the input space differently to the output. Hence they can be made valid only in a local region.

The idea of the weighting functions to model the domain of a local models is took up in the HLAM approach. In Chapter 3 the idea is extended to a coherent framework where a set of options is proposed to handle different aspects of such localized models. This set of methods offers a construction kit to customize HLAMs for specific applications. Different definitions and algorithms are presented that give alternative answers to the above described two basic questions.

Three different domain models are proposed that show different degrees of flexibility in enclosing a local region of the input space. The more flexible the model the more costly is the computational expense for it. In benchmark tests it is shown that with all three domain models comparable results can be achieved. Still, the best compromise between flexibility and ease of parameter estimation offers the so-called hyper-elliptical domain model. It features also a sharp boundary around a domain that allows a clear distinction between its in- and its outside. This makes it possible to detect outliers which are input samples for that an estimation of the correct output can not be guaranteed. This can happen and can be helpful when the available training set does not contain any sample from the region where the outlier comes from. Such an outlier detection mechanism is not possible with approaches from the surveyed literature. Their weighting functions always assign at least one local model to any input sample regardless of the distance between the actual outlier and the model's domain.

Another distinct feature of the HLAM approach are the so-called gating laws. Such a gating law defines to which extend the output of the local models should be mixed in order to compute the output of the whole network. Two alternatives are proposed: either the local models cooperate according to the mixing gating law that has equivalents in the literature or only one local model is exclusively selected to compute the network's output. In experiments it is shown that the latter clearly outperforms the mixing gating law. In different benchmark tests the achieved approximation accuracy improves when only one model defines the output. This works although such an exclusive selection introduces discontinuities in the approximation of the target function. It is quite likely that this contra-intuitive finding is grounded in the proposed learning algorithm that implicitly favors an exclusively selection.

In the HLAM framework any type of supervised machine learning technique can be employed to establish the local models. The learning algorithms for the whole network only selects subsets of samples of the given training set. The chosen MLT has to create local models for these subsets. For this task, simple linear models are most often and successfully used in the experiments described in this thesis. Due to the general definitions of the HLAM approach, even HLAMs itself can be used as local models. Doing so one can built up hierarchies of locally arranged models.

Different information sources can be employed at different levels of such hierarchies.

In the HLAM approach the so-called domain space is differentiated from the model space. The former is the input space for the weighting functions. The latter defines what information is mapped to the output space of the target function. Each level of an HLAM can have a unique domain space. Such a heterogeneous hierarchy that allows a combination of information of different degrees of abstraction has not been discussed in the relevant literature. In Chapter 7 it is demonstrated how this feature is usefully employed. It allows to introduce conveniently a priori knowledge into a solution for a specific learning problem.

To establish an HLAM an offline and an online learning algorithm is proposed. These define the structure of an HLAM i.e. the number of local models and where their domains are located. Both algorithms solve a clustering problem that is constrained by the actual goal to approximate the target function. The main strategy for deciding whether and where a new domain should be inserted is driven by the achieved approximation quality. The desired result is that more local models are inserted in those regions where the target function is more complicated to be approximated. It is advantageous for the application of the HLAM approach that the offline and online algorithm are controlled with just two and three meta-parameters, respectively. In contrast to the commonly used gradient descent approaches that require the tuning of various learning rates the meta-parameters of the HLAM learning algorithms have a clear meaning that facilitates their selection.

Furthermore an algorithm is proposed that can unify domains of an HLAM in order to reduce the number of local models. This can increase the robustness and ease the analysis of a created network. The algorithm is based on a so-called neighborhood graph that ensures that unified domains still adhere to the locality principle. Characteristic for this graph is that it establishes the topology of domains in purely data driven manner. No fixed neighborhood structure has to be manually pre-specified.

All these new definitions and methods are validated in Chapter 4. The various options for the different aspects of an HLAM are compared and conclusions are drawn which option should be favored for what problem. With three public benchmark tests the superiority of the HLAM approach over eight competitors is shown.

Part II: The second topic of this thesis is learning-based robot vision systems. In Chapter 5 the goal is outlined to create highly reactive systems that are adaptive to their environment and can follow long-term goals. It is argued that the weak point of currently available systems is their software. The task-dependent processing of sensory data is still not sufficiently robust and fast. It also remains a problem how information of different levels of abstraction can be combined. Widely accepted is that machine learning methods can simplify the development of robot vision systems. They ease the processing of real-world sensory data and allow a system to learn what action is appropriate in a specific situation.

In Chapter 6 the goals and the software architecture of the COSPAL system are outlined. The challenges of the development of this robot-vision system are to find new learning strategies that allow a combination of continuous and symbolic data processing. To do so, the COSPAL architecture comprises three modules that represent information

processing on different levels of abstraction. These are named "symbolic processing", "grounding-management" and "perception-action". Their names indicate their purpose.

In the perception-action module the HLAM approach is utilized is robot control tasks. The concrete test scenario is a puzzle game where it has to be learned how different blocks can be inserted into matching holes. As described in Chapter 7 a set of HLAMs is developed to realize four basic behaviors needed to play the puzzle. The system can be trained to approach different objects, to align the gripper to them, to grab puzzle blocks and to insert them into holes. The needed HLAMs are distributed in two submodules called pilot and navigator. Their interplay realizes interleaving perception-action cycles that drive the robot w.r.t. visual feedback.

With the pilot a visual servoing algorithm is implemented that offers a simple principle to learn robot movements. It is based on the acquisition of samples how it looks like when the robot moves correctly. After such a bootstrapping phase, an HLAM can be trained to relate the issued robot commands to the resulting changes in the visual appearance of the world. The resulting HLAM can compute robot commands that realize desired changes of the world. It learns to control the robot as it is shown with the samples. By means of the online learning algorithm for HLAMs this visual servoing scheme can be refined during operation. Each performed robot command gives a new training sample how a correct movement looks like. Hence the system can be made robust against changes of the environment. That this online refinement works fast and secure is shown in an experiment where the camera is shifted significantly after the bootstrapping phase.

The pilot and navigator submodules utilize features of the HLAM approach which are not available with generic MLTs. The distinction of the domain and the model space bases a method to extend the competences of a network by simply adding new local models. Such local models are trained for different tasks of the navigator to handle objects specifically (e.g. estimating the optimal pose how a cylindrical block should be grabbed). By employing an object type description as the information that defines the domain space, the HLAMs can learn the characteristics of different blocks and holes. Since these object type descriptions are unambiguous such local models can be established during runtime and extend the networks without interfering the already established functionalities. The other HLAM specific feature, the automatic outlier detection mechanism is used to trigger such competence extensions and to generate failure signals. The former happens for example when an yet unknown object is detected by the image processing methods so that within the navigator submodule a new local model can be trained. On the other hand, failure signals can be send to other modules of the COSPAL architecture when an HLAM is not trained to handle a certain object properly.

As described in Chapter 8 experiments with the implemented COSPAL system prove the functional reliability of the employed HLAMs. Practice shows that HLAMs can be configured quite easily. In all cases the appropriate domain models, the type of used local models and the meta-parameters of the learning algorithms are selected without extensive optimization procedures.

Furthermore, the HLAM approach is used to realize an equivalent of a finite state automaton. This automaton can invoke the four basic behaviors so that the puzzle game is played properly. An HLAM learns the needed transition function that relates game

situations to appropriate action commands. These game situations and commands are given as symbols (i.e. integer). This demonstrates that HLAMs are well suited to handle purely discrete information sources. The trained HLAM also exemplifies how a network can be analyzed w.r.t. its local models and their domains. One can understand that the different states of the automaton correspond to different positions in the domain space, while the local models compute the correct output for each state. With such an analysis it becomes obvious that the realized finite state automaton is correct.

9.2. Outlook

As always, some questions remain open and offer directions for further investigations. In the following some algorithmic problems are considered as well as more general implications of the HLAM approach.

As disclosed with experiments the online learning algorithm for HLAMs does not take full advantage of all presented training samples. A not insubstantial number of samples is not assigned to any local model of a network. Such samples remain unused for the model parameter estimation process. The proposed scheme that assigns new samples to local models in accordance to the neighborhood graph should be revisited. An fruitful idea could be to extend the realized neighborhood concept to more than just two adjacent domains. This could have the desired effect that more samples are found in a local region so that these can be assigned to a new local model.

Another point for improvement concerning the neighborhood graph is its generation. In the thesis only an offline algorithm is proposed that assumes that all samples are available anytime. An incremental version would be better suited for the HLAM online learning algorithm. The problem is that with any processed sample one domain of the network will change (i.e. its position will be shifted) so that the neighborhood relations between all adjacent domains will change, too. Hence an online version of the neighborhood graph generation algorithm has to ensure that the locality principle is not violated.

Likewise important for the online learning algorithm is a characteristic of the domain unification algorithm. As its name indicates it reduces as desired the number of local model by unifying domains. Another reasonable option would be to remove domains and their models in order to realize more compact networks. Especially in the case of target functions that change over time it would be advised to erase local models from regions of the domain space where no new samples will be available anymore. Such useless local models will only increase the computational costs of an HLAM. A first idea to resolve this problem is to utilize the number of recent updates of a local model with new training samples in order to establish a measure for its importance. If too less samples are assigned to a local model it could be removed. The visual servoing algorithm proposed for the COSPAL system with its online refinement mechanism would be a good candidate to test such improvements.

Not sufficiently examined is the poor sparseness of the support vector domain achieved with some of the benchmark tests. The high demand of training samples to establish the domains is surprising as support vector machines commonly create very sparse models. A

closer look at the used kernel function and its meta-parameter might qualify the support vector domain model as a proper alternative to the currently favored hyper-elliptical domain model.

With this thesis the computational complexity of the generation and application of HLAMs remains unexplained. It is left to further research how many steps the recursive offline learning algorithm is likely to need for certain types of data. It is guaranteed that the algorithm will always stop since a finite training set can not be divided arbitrarily often. But there might be a systematic component that explains when more recursion steps are needed. For example one could examine to which extend the approximation error is reduced with each recursion. This could lead to a stop criterion which helps to avoid further divisions of the training set that do not improve the approximation quality. Likewise the costs of computing the output of an HLAM can be analyzed. Experiments with the different domain models w.r.t. this question could be helpful to assess their quality. One can also examine how the process of selecting a local model to compute the output can be speeded up. Since the domains are arranged in a certain topology the input sample has to be compared with each domain individually. The neighborhood graph could be the base of a mechanism that limits significantly the number of candidate models.

Besides such rather practical questions one can start to investigate why the HLAM approach achieves that good benchmark results. The most prominent question is certainly why the discontinuities of the approximation induced by the exclusive gating law does cause any noticed problem. Common approaches realize continuous output functions that smooth the transition between different local models. In contrast to that with a typically configured HLAM, at the border of two domains it is abruptly switched between the corresponding local models. The intuitive conception would be that this decreases the approximation quality. Further research might find that the lack of smoothness is compensated by a higher number of local models. Another reason could be that the applied root mean squared error criterion is not the appropriate mean to disclose disadvantageous jump discontinuities.

The baseline of this thesis is so far: A set of simple linear models that individually approximate a target function in a sharply bounded local region of the input space is a potential mean to solve real-world learning problems.

Bibliography

[1] S. Al-Zubi and G. Sommer. Learning to imitate human movement to adapt to environmental changes. In *Proc. of 18th Int. Conf. on Pattern Recognition (ICPR)*, pages 191–194, Aug. 2006.

[2] S. Al-Zubi and G. Sommer. Learning to mimic motion of human arm and hand grabbing for constraint adaptation. In K. Franke, K. Müller, B. Nickolay, and R. Schäfer, editors, *Pattern Recognition 28th DAGM Symposium*, volume 4174 of *Lecture Notes in Computer Science*, pages 556–565. Springer, Sep. 2006.

[3] J. Aleotti and S. Caselli. Robust trajectory learning and approximation for robot programming by demonstration. *Robotics and Autonomous Systems, Special Issue*, 54(5):409–413, 2006.

[4] S. A. Amari. Dynamics of pattern formation in lateral-inhibition type neural fields. *Biological Cybernetics*, 27:77–87, 1977.

[5] C. Anderson and Z. Hong. Reinforcement learning with modular neural networks for control. In *Proc. of IEEE International Workshop on Neural Networks Applied to Control and Image Processing*, pages 90–93, 1994.

[6] P. Bakker and Y. Kuniyoshi. Robot see, robot do: An overview of robot imitation, http://citeseer.ist.psu.edu/19433.html, 1996.

[7] A. Billard and R. Dillmann, editors. *Robotics and Autonomous Systems, Special Issue: The Social Mechanisms of Robot Programming By Demonstration*, volume 54, 2006.

[8] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[9] R. P. Bonasso. Integrating reaction plans and layered competences through synchronous control. In *Proc. of IJCAI*, pages 1225–1233, 1991.

[10] Cynthia Breazeal, Matt Berlin, Andrew Brooks, Jesse Gray, and Andrea L. Thomaz. Using perspective taking to learn from ambiguous demonstrations. *Robotics and Autonomous Systems, Special Issue*, 54(5):385–393, 2006.

[11] C. Bregler and J. Malik. Learning appearance based models: mixtures of second moment experts. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Proc. in Advances in Neural Information Processing Systems 9*, pages 845–851, 1997.

[12] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, 1984.

[13] J. S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. Fougelman-Soulie and J. Heŕault, editors, *Neuro-computing: Algorithms, Architectures and Applicatations*, pages 227–236. Springer, 1989.

[14] R. A. Brooks. Intelligence without reason. In *Proc. of IJCAI*, pages 569–595, 1991.

[15] Rodney A. Brooks. A robust layered control system for a mobile robot. Technical Report AIM-864, AI Lab, MIT, Sep. 1985.

[16] Rodney A. Brooks. Elephants don't play chess. In P. Maes, editor, *Reasoning about Actions and Plans*, pages 3–16. Elsevier, 1990.

[17] D. S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[18] Thomas Brox, Mikaël Rousson, Rachid Deriche, and Joachim Weickert. Unsupervised segmentation incorporating colour, texture, and motion. In Nicolai Petkov and Michel A. Westenberg, editors, *Computer Analysis of Images and Patterns CAIP*, volume 2756 of *Lecture Notes in Computer Science*, pages 353–360. Springer, Aug. 2003.

[19] J. Bruske. *Dynamische Zellstrukturen - Theorie und Anwendung eines KNN-Modells*. PhD thesis, Institut für Informatik und Praktische Mathematik der Christian-Albrechts-Universität zu Kiel, Report No. 9809, 1998.

[20] J. Bruske and G. Sommer. Dynamic cell structure learns perfectly topology preserving map. *Neural Computation*, 7(4):845–865, 1995.

[21] S. Canu, Y. Grandvalet, and A. Rakotomamonjy. SVM and kernel methods matlab toolbox. Perception Systemes et Information, INSA de Rouen, Rouen, France, 2003.

[22] E. Charniak and D. McDermott. *Introduction to Artificial Intelligence*. Addison-Wesley, Reading, MA, USA, 1987.

[23] Raja Chatila. Deliberation and reactivity in autonomous mobile robots. *Robotics and Autonomous Systems*, 16(2-4):197–211, 1995.

[24] A. Chella, H. Dindo, and I. Infantino. A cognitive framework for imitation learning. *Robotics and Autonomous Systems, Special Issue*, 54(5):403–408, 2006.

[25] K. Chen, D. H. Xie, and H. S. Chi. Speaker identification using time-delay HMEs. *International Journal of Neural Systems*, 7(1):29–43, 1996.

[26] S. Chen, C. F. N. Cowan, and P. M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2:302–309, 1991.

[27] K. B. Cho and B. H. Wang. RBF based adaptive fuzzy systems and their applications to system identification and prediction. *Fuzzy Sets and Systems*, 83:325–339, 1996.

[28] William S. Cleveland and Clive Loader. Smoothing by local regression: Principles and methods. In W. Härtle and M. G. Schimek, editors, *Statistical theory and computational aspects of smoothing*. Physica, 1996.

[29] J. H. Connell. SSS: A hybrid architecture applied to robot navigation. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, Nice, France, May 1992.

[30] COSPAL Consortium. http://www.cospal.org, 2004–2007.

[31] E. R. Davies. *Machine Vision: Theory, Algorithms, Practicalities*. Morgan Kaufmann, 2005.

[32] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Stat. Soc. B*, 39(1):1–38, 1977.

[33] C.L. Blake D.J. Newman, S. Hettich and C.J. Merz. UCI repository of machine learning databases, http://www.ics.uci.edu/∼mlearn/MLRepository.html, 1998.

[34] L Ellis and R Bowden. A generalised exemplar approach to modelling perception action couplings. In *International Workshop on Semantic Knowledge in Computer Vision*, 10th IEEE Int. Conf. Computer Vision, 2005.

[35] Michael Erdmann. Understanding action and sensing by designing action-based sensors. *I. J. Robotic Res*, 14(5):483–509, 1995.

[36] W. Erlhagen, A. Mukovskiy, E. Bicho, G. Panin, C. Kiss, A. Knoll, H. van Schie, and H. Bekkering. Goal-directed imitation for robots: A bio-inspired approach to action understanding and skill learning. *Robotics and Autonomous Systems, Special Issue*, 54(5):353–360, 2006.

[37] B. Espiau, F. Chaumette, and P. Rives. A new approach to visual servoing in robotics. *IEEE Trans. Robotics and Automation*, 8(3):313–326, Jun. 1994.

[38] O. Faugeras. *Three-dimensional Computer Vision*. MIT Press, Cambridge, MA, 1993.

[39] Michael Felsberg, Per-Erik Forssén, Anders Moe, and Gösta Granlund. A COSPAL subsystem: Solving a shape-sorter puzzle. In *AAAI Fall Symposium: From Reactive to Anticipatory Cognitive Embedded Systems*, number FS-05-05 in AAAI Technical Report Series, pages 65–69, Crystal City, USA, Nov. 2005.

[40] Vojtech Franc and Vaclav Hlavac. Simple solvers for large quadratic programming tasks. In Walter G. Kropatch, Robert Sablatnig, and Allan Handbury, editors, *Proc. of German Pattern Recognition Symposium (DAGM)*, volume 3663 of *Lecture Notes in Computer Science*, pages 75–84, Springer, Aug. 2005.

[41] Vojtech Franc and Vaclav Hlavac. A novel algorithm for learning support vector machines with structured output spaces. Research Report K333–22/06, CTU–CMP–2006–04, Department of Cybernetics, Faculty of Electrical Engineering Czech Technical University, May 2006.

[42] Freund and Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55, 1997.

[43] Jürgen Fritsch, Michael Finke, and Alex Waibel. Context-dependent hybrid HME/HMM speech recognition using polyphone clustering decision trees. In *Proc. of IEEE Inter. Conf. on Acoustics, Speech, and Signal Processing*, 1997.

[44] E. Gat. *Reliable Goal-directed Reactive Control for Real-World Autonomous Mobile Robots*. PhD thesis, Virginia Polytechnic and State University, Blacksburg, USA, 1991.

[45] E. Gat. On three-layer architectures. In R. P. Bonnasso D. Kortenkamp and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*. MIT Press, 1997.

[46] J.H. Goodband, O.C.L. Haas, and J.A. Mills. A mixture of experts committee machine to design compensators for intensity modulated radiation therapy. *Pattern Recognition*, 39:1704–1714, 2006.

[47] G. H. Granlund. An associative perception-action structure using a localized space variant information representation. In G. Sommer and Y. Zeevi, editors, *2nd Int. Workshop on Algebraic Frames for the Perception-Action Cycle*, volume 1888 of *Lecture Notes in Computer Science*, pages 48–68. Springer, 2000.

[48] G. H. Granlund. A Cognitive Vision Architecture Integrating Neural Networks with Symbolic Processing. *Künstliche Intelligenz*, (2):18–24, 2005. Böttcher IT Verlag, Bremen, Germany.

[49] G. H. Granlund, G. Sommer, J. Kittler, and V. Hlavac. The COSPAL Project. *6th Framework Programme: Information Society Technologies*, IST-2003-2.3.2.4 Cognitive Systems, Project Number: IST-004176.

[50] Gösta Granlund. *Cognitive Vision Systems*, chapter Organization of Architectures for Cognitive Vision Systems, pages 37–55. Springer, Heidelberg, 2006. Hans Hellmut Nagel and Henrik I. Christensen eds.

[51] G. Hailu. *Towards Real Learning Robots*. PhD thesis, Institut für Informatik und Praktische Mathematik der Christian-Albrechts-Universität zu Kiel, Report No. 9906, 1999.

[52] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, 2001.

[53] D. O. Hebb. *The organisation of behavior*. Wiley, New York, 1949.

[54] J. Hill and W. T. Park. Real time control of a robot with a mobile camera. In *Proc. 9th ISIR*, pages 233–246, Mar. 1979.

[55] F. Hoppe and G. Sommer. Local linear models for system control. In *Proc. of Int. Conf. on Neural Information Processing (ICONIP)*, pages 171–176, 2005.

[56] F. Hoppe and G. Sommer. Ensemble learning for hierarchies of locally arranged models. In *Proc. of IEEE World Concress on Computational Intelligence*, pages 10612–10619, 2006.

[57] F. Hoppe and G. Sommer. Fusion algorithm for locally arranged linear models. In *Proc. of 18th Int. Conf. on Pattern Recognition (ICPR)*, pages 1208–1211, 2006.

[58] F. Hoppe and G. Sommer. Online learning for hierarchical networks of locally arranged models using a support vector domain model. In *Proc. of Int. Joint Conf. on Neural Networks (IJCNN)*, to be published, 2007.

[59] Guang-Bin Huang, Paramasivan Saratchandran, and Narasimhan Sundararajan. An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 34(6):2284–2292, 2004.

[60] S. A. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. *IEEE Trans. Robotics and Automation*, 12(5):651–670, Oct. 1996.

[61] R. Jacobs, M. Jordan, S. J. Nowlan, and Hinton Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

[62] Robert A. Jacobs, Michael I. Jordan, and Andrew G. Barto. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. *Cognitive Science*, 15(2):219–250, 1991.

[63] Erik Jonsson, Michael Felsberg, and Gösta Granlund. Incremental associative learning. Technical Report LiTH-ISY-R-2691, Dept. EE, Linköping University, Sweden, Sept. 2005.

[64] Michael I. Jordan and R. A. Jacobs. Hierarchial mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

[65] B. Julesz. Early vision is bottom-up, except for focal attention. *Cold Spring Harb Symp Quant Biol*, 55:973–978, 1990.

[66] V. Kadirkamanathan and M. Niranjan. A function estimation approach to sequential learningwith neural networks. *Neural Computation*, 5(6):954–975, 1993.

[67] Y. Kassahun. *Towards a Unified Approach to Learning and Adaptation.* PhD thesis, Institut für Informatik und Praktische Mathematik der Christian-Albrechts-Universität zu Kiel, Report No. 0602, 2006.

[68] Loo Chu Kiong, Mandava Rajeswari, and M. V. C. Rao. Extrapolation detection and novelty-based node insertion for sequential growing multi-experts network. *Neural Network World*, 2:151–176, 2003.

[69] S. Kirstein, H. Wersing, and E. Körner. Rapid online learning of objects in a biologically motivated recognition architecture. In *Proc. of German Pattern Recognition Symposium (DAGM)*, volume 3663 of *Lecture Notes in Computer Science*, pages 301–308. Springer, 2005.

[70] Josef Kittler, William J. Christmas, Alexey Kostin, Fei Yan, Ilias Kolonias, and David Windridge. A memory architecture and contextual reasoning framework for cognitive vision. In *14th Scandinavian Conference Image Analysis (SCIA)*, volume 3540 of *Lecture Notes in Computer Science*, pages 343–358. Springer, 2005.

[71] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[72] T. Kohonen. *Self-Organizing Maps*. Springer, Berlin, 1995, 1997, 2001.

[73] D. Kragic and H. Christensen. Survey on visual servoing for manipulation. Technical Report CVAP259, Jan. 2002.

[74] Xiaoping Lai and Bin Li. An efficient learning algorithm generating small RBF neural networks. *Neural Network World*, 15:525–533, 2005.

[75] L. Ljung and T. Soderstrom. *Theory and practice of recursive identification*. MIT Press, 1983.

[76] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the 5th Berkeley Symposium on Mathematics*, volume 1 of *Statistics and Probability*, pages 281–296, 1967.

[77] Marvin Minsky. *The Society of Mind*. Simon and Schuster, 1985.

[78] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, *1*(2):281–294, 1989.

[79] Jun Nakanishi, Jay A. Farrell, and Stefan Schaal. A locally weighted learning composite adaptive controller with structure adaptation. In *IEEE/RSJ Inter. Conf. on Intelligent Robots and Systems*, pages 882–889, 2002.

[80] Jun Nakanishi, Jay A. Farrell, and Stefan Schaal. Composite adaptive control with locally weighted statistical learning. *Neural Networks*, 18(1):71–90, 2005.

[81] A. Newell and H. Simon. GPS: A program that simulates human thought. In Feigenbaum and Feldman, editors, *Computers and Thought*, pages 279–293. McGraw Hill, 1963.

[82] Nils J. Nilsson. *Principles of Artificial Intelligence*. Tioga, 1980.

[83] Masaki Oginoa, Hideki Toichia, Yuichiro Yoshikawaa, and Minoru Asada. Interaction rule learning with a human partner based on an imitation faculty with a simple visuo-motor mapping. *Robotics and Autonomous Systems, Special Issue*, 54(5):414–418, 2006.

[84] E. J. Ong and R. Bowden. Learning distances for arbitrary visual features. In *British Machine Vision Conference*, page II:749, 2006.

[85] J. Pauli. *Learning-Based Robot Vision*, volume 2048 of *Lecture Notes in Computer Science*. Springer, 2001.

[86] Axel Pinz. Object categorization. *Foundations and Trends in Computer Graphics and Vision*, 1(4):255–353, 2006.

[87] J. Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3(2):213–225, 1991.

[88] J. Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3(2):213–225, 1991.

[89] John C. Platt, Bernhard Schölkopf, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. Technical Report MSR-TR-99-87, Microsoft Research, Nov. 1999.

[90] H. Prehn and G. Sommer. An adaptive classification algorithm using robust incremental clustering. In *Proc. of 18th Int. Conf. on Pattern Recognition (ICPR)*, pages 896–899, Aug. 2006.

[91] H. Prehn and G. Sommer. Incremental classifier based on a local credibility criterion. In V. Devedzic, editor, *Proc. of IASTED Int. Conf. on Artificial Intelligence and Applications (AIA)*, pages 372–377, Feb. 2007.

[92] J. Ross Quinlan. Combining instance-based and model-based learning. In *Proc. of ICML*, pages 236–243, 1993.

[93] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, USA, 1993.

[94] V. Ramamurti and J. Ghosh. Structural adaptation in mixture of experts. In *Int. Conf. on Pattern Recognition*, volume 4, pages 704–708, 1996.

[95] D. Brian Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[96] H. Ritter, T. Martinetz, and K. Schulten. *Neuronale Netze*. Addison Wesley, 2th edition, 1991.

[97] Helge Ritter. Learning with the self-organizing map. In T. Kohonen, editor, *Artificial Neural Networks*, pages 379–384. Elsevier, 1991.

[98] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Science*, 3(6):233–242, Jun. 1999.

[99] Stefan Schaal and Christopher G. Atkeson. Constructive incremental learning from only local information. *Neural Computation*, 10(8):2047–2084, 1998.

[100] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[101] Amanda J. C. Sharkey. Types of multinet system. In *Proc. Int. Workshop on Multiple Classifier Systems*, volume 2364 of *Lecture Notes in Computer Science*, pages 108–117. Springer, Jun. 2002.

[102] Amanda J.C. Sharkey, editor. *Combining artificial neural nets: ensemble and modular multi-net systems*, chapter Mixtures of X, pages 267–295. Springer, 1999.

[103] Robert Shorten and Roderick Murray-Smith. Side-effects of normalising basis functions in local model networks. In Roderick Murray-Smith and Tor Arne Johansen, editors, *Multiple Model Approaches to Modelling and Control*, chapter 8, pages 211–228. Taylor & Francis, 1997.

[104] G. Sommer. Algebraic aspects of designing behavior based systems. In G. Sommer and J.J. Koenderink, editors, *Algebraic Frames for the Perception and Action Cycle*, volume 1315 of *Lecture Notes in Computer Science*, pages 1–28. Springer, 1997.

[105] Jochen J. Steil, Frank Röthling, Robert Haschke, and Helge Ritter. Situated robot learning for multi-modal instruction and imitation of grasping. *Robotics and Autonomous Systems*, 47(2-3):129–141, 2004.

[106] Michael Tagscherer, Lars Kindermann, Achim Lewandowski, and Peter Protzel. Overcome neural limitations for real world applications by providing confidence values for network prediction. In *Proc. of Int. Conf. on Neural Information Processing (ICONIP)*, pages 520–525, 1999.

[107] Bin Tang, Malcom I. Heywood, and Michael Shepherd. Input partitioning to mixture of experts. In *Int. Joint Conf. on Neural Networks (IJCNN)*, pages 227–232, May 2002.

[108] Sebastian Thrun. Probabilistic algorithms in robotics. *AI Magazine*, 21(4):93–109, 2000.

[109] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary

Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. Stanley: The robot that won the darpa grand challenge: Research articles. *J. Robot. Syst.*, 23(9):661–692, 2006.

[110] J. K. Tsotsos. On behaviorist intelligence and the scaling problem. *Artifical Intelligence*, 75(2):135–160, 1995.

[111] S. R. Waterhouse and A. J. Robinson. Classification using hierarchical mixtures of experts. In *Proc. of IEEE Workshop on Neural Networks for Signal Processing*, pages 177–186. IEEE Press, 1994.

[112] L. E. Weiss, A. C. Sanderson, and C. P. Neuman. Dynamic sensor-based control of robots with visual feedback. *IEEE Trans. Robotics and Automation*, 3:404–417, 1987.

[113] Tomáš Werner. A linear programming approach to max-sum problem: a review. Technical Report CTU–CMP–2005–25, Center for Machine Perception, Czech Technical University, Dec. 2005.

[114] W. S. B. Woolhouse. Explanation of a new method of adjusting martality tables, with some observations upon Mr. Makeham's modification of Gompertz's theory. *J. Inst. Act.*, 15:389–410, 1870.

[115] Lu Yingwei, N. Sundararajan, and P. Saratchandran. A sequential learning scheme for function approximation using minimal radial basis function neural networks. *Neural Computation*, 9(2):461–478, 1997.

Glossary

CD	Center Domain (Model)
DCS	Dynamic Cell Structures
EG	Exclusive Gating (Law)
GM	Grounding-Management (Module)
HED	Hyper-elliptical Domain (Model)
HLAM	Hierarchical Network of Locally Arranged Models
MG	Mixing Gating (Law)
MLT	Machine Learning Technique
PA	Perception-Action (Module)
PCA	Principle Component Analysis
RBF	Radial Basis Function
SOM	Self-Organizing Map
SP	Symbol Processing (Module)
SVD	Support Vector Domain (Model)