DIPLOMARBEIT

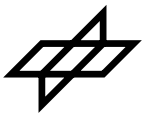
Model-Based Pose Estimation of 3-D Objects from Camera Images Using Neural Networks

Stefan Winkler

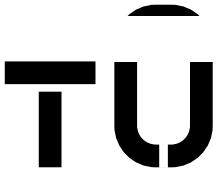
Bahnstraße 2
A-3712 Maissau

BETREUER:

Prof. Dr. Gerd Hirzinger; Patrick Wunsch, M.Sc.
Institut für Robotik und Systemdynamik
DLR Forschungszentrum Oberpfaffenhofen



Deutsche Forschungsanstalt für Luft- und Raumfahrt e.V.



Prof. Dr. Wolfgang Mecklenbräuker
Institut für Nachrichtentechnik und Hochfrequenztechnik
Technische Universität Wien

Oberpfaffenhofen, Juli 1996

Traveller, there is no path.
Paths are made by walking.

Theme of Latin American folk song

Abstract

Machines need the ability to determine the pose of objects in their environment in order to be able to reliably and intelligently interact with them. This thesis investigates neural network approaches to model-based object pose estimation from camera images. Kohonen maps and some of their variations are studied for this purpose. It is shown that the performance of these networks depends heavily on the mathematical representation of pose in general and of orientation in particular. Furthermore, it is most advantageous to choose the topology of the neural network in harmony with the representation employed. The results obtained with simulated as well as real world input images are described, and a detailed analysis of the influence of various parameters and external conditions is given

Kurzfassung

Ohne die Fähigkeit, die Lage von Objekten zu erkennen, können Maschinen nicht intelligent und zuverlässig mit ihrer Umwelt interagieren. Diese Diplomarbeit untersucht das Problem der modellbasierten Lageschätzung aus Kamerabildern mit Hilfe von neuralen Netzwerken. Kohonen-Karten und einige ihrer Abwandlungen werden in dieser Hinsicht auf ihre Eignung getestet. Es wird gezeigt, daß die Leistung dieser neuralen Netzwerke hier sehr stark von der mathematischen Darstellung der Objektlage im allgemeinen und der Rotation im speziellen abhängt. Ebenso ist es von Vorteil, eine auf diese mathematische Darstellung abgestimmte Topologie für das neurale Netzwerk zu verwenden. Eine Beschreibung der Ergebnisse sowohl mit simulierten als auch mit realen Kamerabildern sowie eine detaillierte Analyse des Einflusses verschiedener Parameter und externer Bedingungen bilden den Abschluß.

Acknowledgements

This thesis is the result of research conducted with the Vision Group of the Institute of Robotics and System Dynamics at the Research Center Oberpfaffenhofen of the German Aerospace Research Establishment (Deutsche Forschungsanstalt für Luft- und Raumfahrt). Thanks to Prof. Dr. Gerd Hirzinger, the director of the institute, for giving me the opportunity to work in this stimulating environment with its outstanding facilities.

I am indebted to Patrick Wunsch, my supervisor at the DLR, for his continuous support and guidance, for many fruitful discussions and ideas, and last but not least for his assistance with Datacube implementations.

Furthermore I want to thank Jörg Langwald, Dr. Klaus Arbter and Gernot Koegel, who have contributed valuable suggestions along the way, as well as people around the world on the internet too numerous to mention here, who compiled a wealth of helpful information to make it available on-line or have responded directly to my queries.

I am grateful to Prof. Dr. Wolfgang Mecklenbräuker at the Department of Communication and Radio-Frequency Engineering of the University of Technology in Vienna, who readily agreed to supervise my thesis there.

Special thanks go to Ana-Maria Lupas for her diligence in hunting down English imperfections in this paper.

This research was made possible in part by a Kurt Gödel Research Grant from the Austrian Ministry of Science, Research and Arts.

Contents

"The time has come," the walrus said,
"To talk of many things."

Lewis Carroll's Tweedledee

1 Introduction

Robots are employed in widely varying areas today, and many fields of applications rise up where the use of robots would have been unimaginable a few years back. These achievements are mainly based on new kinds of control mechanisms and an adaptation of robot interfaces to human behavior and needs.

Many of the problems that arise in robotics are caused by the limited ability of robots to react to changes in their environment and by their inflexibility in coping with noisy inputs. This is often aggravated by considerable delays in command processing as well as unnatural and insufficient human-machine interfaces with utterly limited bandwidth. One way to overcome these impediments is making robots more independent and self-contained by giving them the ability to solve complex problems in their environment using the feedback from a multitude of sensors. Commands can then be issued on a higher level and naturally allow for a better interaction with humans.

In order to achieve a greater flexibility it is essential that the robot be able to determine its own orientation and position with respect to its surroundings. Furthermore the robot must be able to detect, track, grasp and manipulate free-moving objects that are not part of a fixed environment.

Prior to any kind of manipulation it is necessary to recognize the object and to determine (and keep track of) its pose in space. This is performed in several sequentially executable stages, which are depicted in Figure 1-1. First comes segmentation, which comprises the necessary preprocessing of sensor data as well as the generation of features relevant to the task at hand. It can be done on the basis of intensity, color, texture or distance. Afterwards, one or more hypotheses about the object and its approximate pose are generated only to be either rejected or confirmed and refined in the ensuing verification stage. Once the object and its pose are determined, the object can be tracked in real time with a precision sufficient for the control of the robot [48].

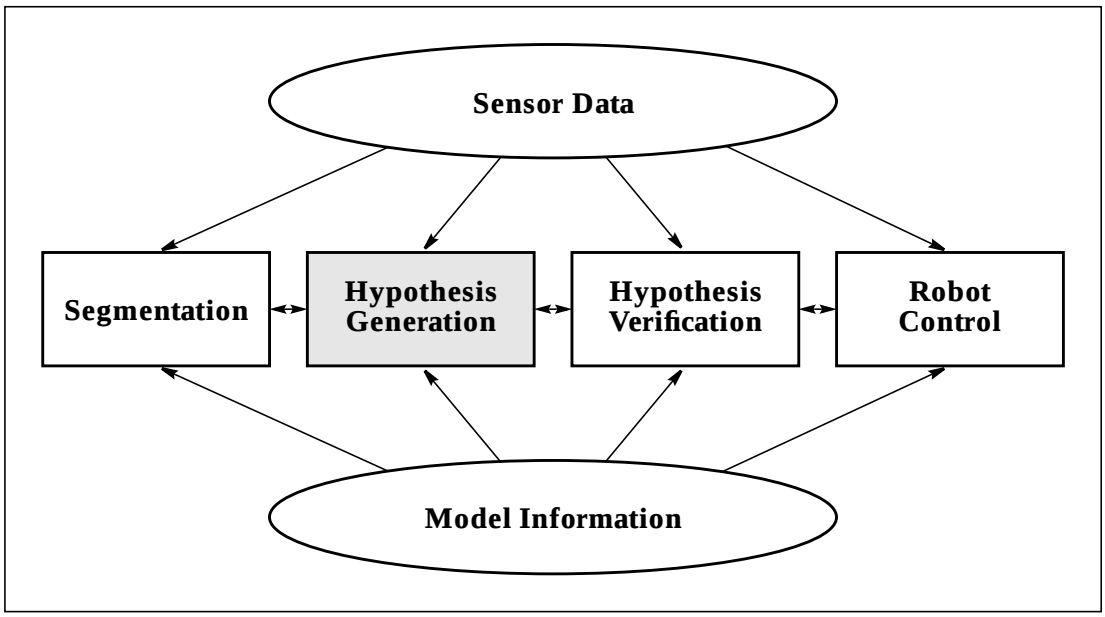


Figure 1-1: Stages of multi-sensor robot control [63]

Throughout these stages, a wide range of sensor data can be made available. Visual information from cameras is easily obtained and therefore obvious starting material for any kind of environment analysis. Laser triangulation can be employed to provide depth profiles, and tactile sensors could provide surface normals of the object. In addition to sensor data, a priori knowledge about the object such as color, texture and geometry, is modelled in the computer to support the calculations at every stage.

Considering the human visual system, interpreting three-dimensional objects should be feasible from a single two-dimensional image without the presence of depth data. However, several major difficulties need to be coped with, since a considerable amount of data is irretrievably lost in the perspective projection from 3D to 2D performed by the camera. This projection is not only responsible for the fact that even simple objects exhibit an enormous number of different aspects when viewed from varying positions, but it also produces highly discontinuous output due to the sudden disappearance or emergence of surfaces and edges of the object even at minor changes of the viewing position. Furthermore, the perspective projection is not invariant with respect to translations. All this complicates the determination of the correspondence between image and model features.

Numerical feature matching methods constitute an efficient way of iteratively refining an initial estimation of the object's pose to a precision theoretically limited only by the resolution of the camera. In general, these algorithms are composed of two successive steps, the first of which determines correspondences between image features and model data, while the second computes the rigid body transformation that minimizes the displacement of matched features [62]. However, such methods are of no avail unless a reasonable estimate of the initial pose can be given.

Multi-view representations combined with methods of artificial intelligence are a viable solution for determining this initial pose approximation. Object features are extracted from the camera image and compared to the model features stored in a database containing all

possible aspects of the object. This database can be represented by an aspect graph or tree, which in the process is searched for the aspect best matching the current view. However, the complexity of the search grows exponentially with the number of features under consideration. Not even the use of clever search algorithms can rule out extraordinary memory and computing power requirements, which essentially disqualifies multi-view representations for real-time applications. Langwald [32] studies these approaches and their applicability in detail.

Classification-based methods that infer the object's pose from two-dimensional feature patterns present themselves as an alternative to multi-view representations. Due to the complexity of the underlying perspective projection, classifiers based on neural networks appear particularly promising. Neural networks have proven to be versatile function approximators even for multi-dimensional data and can be trained off-line, which constitutes a tremendous performance advantage under real time conditions.

This thesis investigates automatic model-based pose estimation of three-dimensional objects from two-dimensional camera images using neural networks. The ultimate goal is a combination of image processing steps with a neural network that has the ability to generate in real time a reliable pose estimate of an object. This estimate need only be precise to such a degree that it comes to lie within the region of convergence of the feature matching or tracking algorithm used subsequently to determine the accurate pose.

In chapter 2, "Neural Networks", an overview of neural network fundamentals is given and various possible network structures and training algorithms as well as their applicability to the problem at hand are discussed. In chapter 3, "Pose Representations", we elaborate on suitable mathematical representations of an object's pose and design appropriate network topologies. Chapter 4, "Object Features", deals with the selection of suitable object features for this task, how they can be modelled, and how they may be extracted from the camera image. In chapter 5, "Results", we evaluate the experiments that were carried out and present the results obtained. The discussion in chapter 6 finally recapitulates the methods under investigation and their applicability to this problem, and it also attempts to provide motivation and ideas for further research.

The scientist does not study nature because it is useful; he studies it because he delights in it, and he delights in it because it is beautiful. If nature were not beautiful, it would not be worth knowing, and if nature were not worth knowing, life would not be worth living.

Henri Poincaré

2 Neural Networks

Artificial neural network models have been studied for many years in the hope of achieving human-like performance in the broad field of pattern recognition and classification. The development of mathematical models began more than 50 years ago with the work of McCulloch, Pitts, Hebb, Rosenblatt, Widrow and others, and many different models and algorithms have evolved since. For an overview of past and present research efforts the reader is referred to [3], [33] and [60].

Generally speaking, neural networks are universal function approximators [19][57]. They present themselves as a useful alternative to “classical” methods in either of the following two cases:

- The function to be approximated is unknown.
- The function to be approximated is known, but determining its parameters is difficult or impossible.

The former holds true in the case of pose estimation due to the complexity and discontinuity of the perspective projection transformation of opaque objects.

We concentrate on competitive learning and its self-organizing implementation, the Kohonen feature map. After a description of these general concepts and their behavior, several related modifications and enhancements will be discussed. Finally, previous research efforts in the field of pose estimation with neural networks are reviewed.

2.1. The Neuron

Artificial neural networks attempt to model the function of the brain. A typical biological neuron has several thousand incoming nerves or dendrites connected to it via synapses and one outgoing nerve or axon extending from it. The influence of each dendrite is determined by the strength of its synapse. The neuron “fires” when the total weighted input from all dendrites exceeds a certain threshold. The massive parallelism inherent in the operation of the brain outweighs the comparatively low processing speed of the neurons.

A simple model of the neuron is shown in Figure 2-1. With its M weights $w_1, \ldots, w_M$, the neuron forms a weighted sum of its M inputs $\hat{w}_1, \ldots, \hat{w}_M$ and passes the result through some kind of nonlinearity:

$$y = f\left(\sum_{j=1}^{M} w_j \cdot \hat{w}_j\right). \tag{2-1}$$

This nonlinearity can be a discrete threshold function or a continuous function. From the point of view of function approximation, (2-1) can be interpreted in analogy to Fourier series, for instance, which approximate functions similarly, albeit as a *linear* combination:

$$y = \sum_{j} w_j \cdot f_j(\hat{w}_j). \tag{2-2}$$

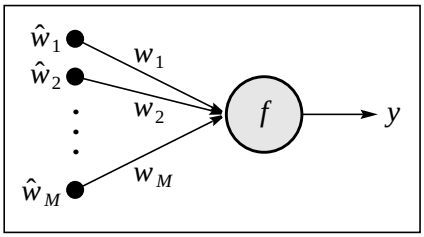


Figure 2-1: A simple model of a neuron [19]. The output y of the neuron is a nonlinear function of the weighted sum of its inputs and may be calculated according to (2-1).

2.2. The Kohonen Feature Map

Feed-forward networks trained with the backpropagation algorithm seem to be very popular in the neural network community, even though they tend to be rather difficult to handle for a variety of reasons [14][18]:

- Choosing the number of hidden layers and the number of neurons within every one of them requires a lot of experience.
- The convergence properties of the backpropagation algorithm heavily depend on the initial weights and the training parameters.
- The backpropagation algorithm may require a long time for training.
- The interpretation of weight values is difficult due to the distributed character of information storage in the network.

The Kohonen feature map and its relatives try to overcome these problems. They are based on the concept of competitive learning and implement a generalized self-organizing process that facilitates the automatic formation of topologically correct mappings [25].

2.2.1. Competitive Learning

Competitive learning is a process in which the nodes of a neural network are tuned to specific input patterns by "competing" with each other during training. Assume a network consisting of a single layer of N neurons or output nodes, each of which is fully connected to a set of M inputs. Each node i stores a (randomly initialized) weight vector $\mathbf{w}_i = [w_{i1}, w_{i2}, \ldots, w_{iM}]$. At every time instant t an input vector $\hat{\mathbf{w}} = [\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_M]$ is presented to the network. Based on some distance measure (see below), the best-matching node w (the *winner*) is selected and updated in such a way that the distance between its weight vector $\mathbf{w}_w$ and the input vector $\hat{\mathbf{w}}$ is decreased by a certain fractional amount λ, the learning rate:

$$\Delta \mathbf{w}_w = \lambda(t) \cdot (\hat{\mathbf{w}} - \mathbf{w}_w) . \tag{2-3}$$

This is the so-called *winner-takes-all rule*. In the long run, the nodes tend to become tuned to different domains of the input.

In his book [26], Kohonen gives a rather detailed overview of possible distance or similarity measures. Among the ones mentioned are *correlation*

$$C_i = \mathbf{w}_i \cdot \hat{\mathbf{w}} = \sum_{j=1}^{M} w_{ij} \hat{w}_j , \tag{2-4}$$

which is equivalent to the scalar or inner product if $\mathbf{w}_i$ and $\hat{\mathbf{w}}$ are understood as vectors in Euclidean space, the *direction cosine*

$$\cos \varphi_i = \frac{\mathbf{w}_i \cdot \hat{\mathbf{w}}}{\|\mathbf{w}_i\| \|\hat{\mathbf{w}}\|} \tag{2-5}$$

measuring the angle between two vectors, both of which are similarity measures, and the *Euclidean distance* measure

$$\delta_i = \|\mathbf{w}_i - \hat{\mathbf{w}}\| . \tag{2-6}$$

Obviously the results obtained using these methods are identical if the input and weight vectors are normalized. In this case, the winner w can be determined by finding either the node with the largest scalar product of its weight vector and the input vector

$$\mathbf{w}_w \cdot \hat{\mathbf{w}} \geq \mathbf{w}_i \cdot \hat{\mathbf{w}} \quad \forall i , \tag{2-7}$$

or the node with the smallest Euclidean distance from the input

$$\|\mathbf{w}_w - \hat{\mathbf{w}}\| \leq \|\mathbf{w}_i - \hat{\mathbf{w}}\| \quad \forall i . \tag{2-8}$$

2.2.2. Self-Organization

Kohonen presented an elegant way of incorporating self-organization into competitive learning [25]. He introduced a spatial relationship between the nodes by extending the update of the winner's weight vector (2-3) to the "neighbors" of the winner. This can be achieved by assigning a position $\boldsymbol{p}_i$ in some space $\boldsymbol{P}$ to each node i, that determines the topology of the

network and the neighborhood relations of the nodes. The update rule, which now must be carried out for all nodes, is modified to

$$\Delta \mathbf{w}_i = \nu(\boldsymbol{p}_w, \boldsymbol{p}_i, t) \cdot \lambda(t) \cdot (\hat{\mathbf{w}} - \mathbf{w}_i) \; . \tag{2-9}$$

The neighborhood function ν influences the update of the weight vector in such a way that nodes far away from the winner in space $\boldsymbol{P}$ experience less weight change than nodes close to the winner. This procedure makes sure that neighboring nodes will respond to similar inputs and thereby induces the automatic formation of a topologically correct mapping from the input space to $\boldsymbol{P}$ that preserves neighborhood relations.

The choice of the neighborhood function can significantly influence the convergence properties – especially the speed of convergence – of the self-organizing feature map, which is demonstrated for a one-dimensional feature map in [7]. While the particular function chosen is not critical, the best function to use seems to be convex over a large range around the winner and yet have noticeably different values at distant neurons. These competing requirements are nicely balanced in the Gaussian function:

$$\nu(\boldsymbol{p}, \boldsymbol{q}, t) = \nu_0 \cdot e^{-\frac{\delta(\boldsymbol{p}, \boldsymbol{q})^2}{2\sigma^2(t)}} \; . \tag{2-10}$$

The distance $\delta(\boldsymbol{p}, \boldsymbol{q})$ between the two positions $\boldsymbol{p}$ and $\boldsymbol{q}$ depends on the actual topology of $\boldsymbol{P}$ and will be defined in chapter 3, “Pose Representations”, where different representations of pose together with the pertinent network topologies are introduced.

It is found that convergence times are minimal when the spread σ of the neighborhood function is of the order of the extension of the network. Different choices of parameters and functions – particularly concave ones – often lead to “metastable” states, in which the algorithm may be trapped for a long time before converging on the optimal representation [7].

It has proven advantageous to decrease both the learning rate λ and the spread σ of the neighborhood function as training time progresses and the map becomes ordered. In the beginning, when λ and σ are large, many nodes participate substantially in the learning process, leading to a quick coarse ordering of the map. Towards the end of the training, when λ and σ are rather small, only the nearest neighbors of the winner will be noticeably influenced, and the finer details of the mapping are learned. Having both λ and σ decay exponentially with time works fine:

$$\lambda(t) = \lambda(0) \cdot e^{-k_\lambda \cdot t}, \tag{2-11}$$

$$\sigma(t) = \sigma(0) \cdot e^{-k_\sigma \cdot t}. \tag{2-12}$$

A multitude of experiments with higher animals and humans show that self-organization indeed has biological foundations [1]. Especially the visual and the somatosensory system appear to self-organize at least their finer structure only under the influence of external stimuli. The field of vision, for instance, is mapped onto the primary visual cortex preserving neighborhood relations (retinotopic map), the neurons in the auditory cortex are ordered from the lowest to the highest pitch they respond to (tonotopic map), and the somtatotopic map of the skin surface or the motor map are ordered representations of the body. Even abstract

information like categories and semantic values of words seems to be organized in this fashion. Regions of higher interest are generally represented by a greater number of neurons, creating a higher sensitivity and resolution for stimuli from these areas.

2.2.3. Behavior

Kohonen investigates various topologies and their behavior in [26] and [27]. To give an example of the ordering process and of the implementation, assume a two-dimensional input space, i.e. $\hat{\mathbf{w}} = [\hat{w}_1, \hat{w}_2]$ and $\mathbf{w}_i = [w_{i1}, w_{i2}]$. Assume further 16x16 nodes arranged in two-dimensional Euclidean space $\boldsymbol{P}$ with rectangular neighborhood topology, i.e. every node – except for the ones at the borders – has four nearest neighbors. The position of each node in $\boldsymbol{P}$ is given by another two-dimensional vector $\mathbf{p}_i = [p_{i1}, p_{i2}]$, and the distance δ between two nodes is of course calculated as the Euclidean distance

$$\delta(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\| . \tag{2-13}$$

During training, random vector samples are used as input vectors. The probability density function for the coefficients of the input vectors is chosen to be uniform over the entire square area outlined in Figure 2-2. λ is decreased exponentially with time according to (2-11) from 0.9 to 0.02, and likewise σ is reduced with time according to (2-12) from half the side length of the square to 2% of the side length over training 10000 steps.

Figure 2-2 shows the arrangement of the map at different stages of training: The weights of the nodes are initialized with random values ignoring neighborhood relations. After presenting 1000 input vectors to the network, the weight vectors of the nodes have assumed an ordering roughly resembling the internal neighborhood relations. After 10000 training steps not only the neighborhood relations have become perfect, but the weight vectors are also spread almost uniformly over the entire square area.

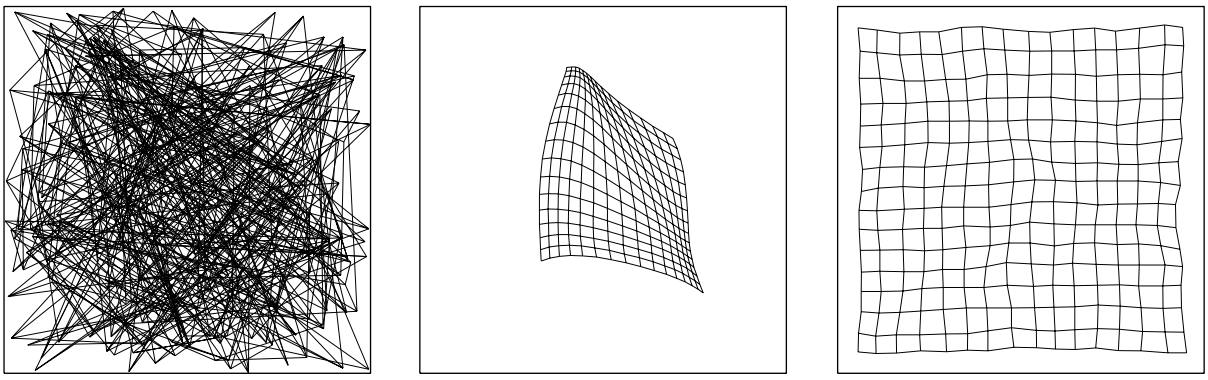


Figure 2-2: Two-dimensional map after 0, 1000, and 10000 training steps with input vectors uniformly distributed across the outlined square area. At every intersection of lines there is a node, whose position in the square is determined by its weight vector. The lines themselves connect each node to at most four of its nearest neighbors in $\boldsymbol{P}$.

In fact, the distribution of the weight vectors represents the probability density function of the input vectors. When four times as many input samples stem from the gray square area in the center of Figure 2-3 than from the surrounding area, for example, more nodes will assume positions within that area, providing a higher resolution for more probable stimuli. In effect the network resources are always used optimally with need. The self-organizing map need not even be restricted to the geometry its nodes are arranged in, even though this will generally introduce some distortion (see Figure 2-3).

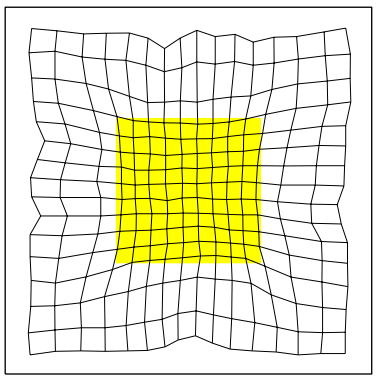


Figure 2-3: Two-dimensional map after training with a non-uniform probability density function. Four times as many input samples stem from the gray square area in the center than from the surrounding area. This leads to a higher density of nodes within that area.

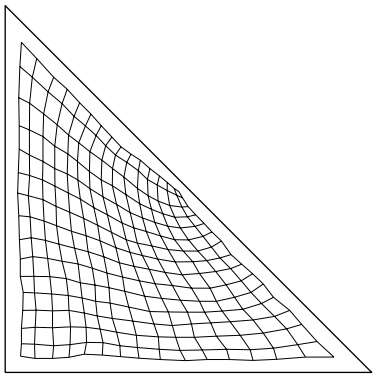


Figure 2-4: Two-dimensional map after training with input vectors uniformly distributed across the outlined triangular area. Again the nodes arrange themselves so as to cover the given region of input vectors observing neighborhood relations. Because the internal rectangular topology of the network is not optimal for triangular areas, notable distortion occurs.

Another interesting property is the automatic selection of feature dimensions by the self-organizing map. If, for instance, input vectors uniformly distributed on a two-dimensional square area are presented to an open one-dimensional “chain” of 256 neurons with two neighbors per node, their weight vectors will meander across the entire area in a way reminiscent of Peano curves[†], attempting to cover the entire region of input vectors as

[†] A Peano curve “fills” a plane, i.e. given some patch of the plane, the curve meets every point in that patch.

demonstrated in Figure 2-5. Comparable effects can be observed for higher-dimensional input spaces and network topologies. Notice that now similar input vectors may well excite nodes far apart in the chain.

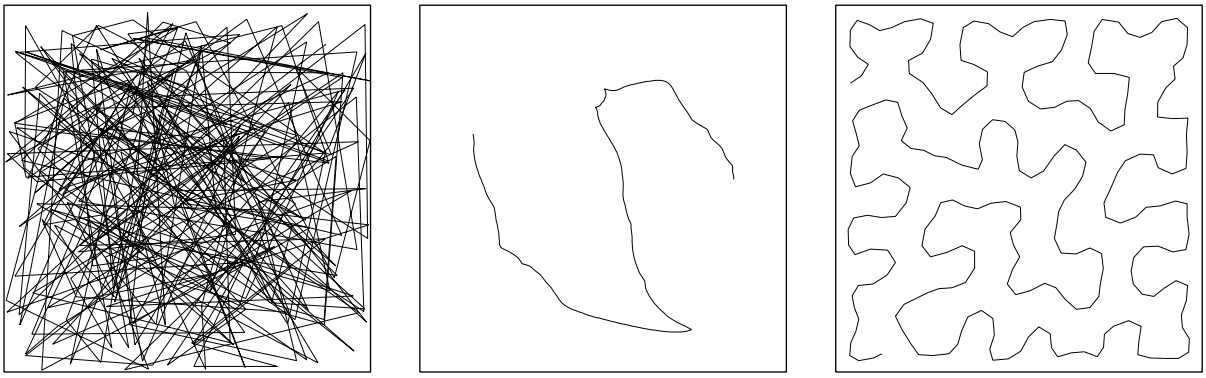


Figure 2-5: One-dimensional map after 0, 100 and 100000 training steps with input vectors uniformly distributed across the square area. Again the nodes arrange themselves so as to cover the given region of input vectors observing neighborhood relations. However, due to the difference in dimension, similar input vectors may well excite nodes far apart in the chain.

2.3. Nodes with Responses

Ritter and Schulten extend Kohonen's feature map by storing a response $\boldsymbol{r}_i$ in every node i, thereby making the topology of the response space $\boldsymbol{R}$ independent of the network's internal topology [52]. These responses are trained in much the same way as the weight vectors; however, each input vector $\hat{\mathbf{w}}$ is now "tagged" with the correct response $\hat{\boldsymbol{r}}$. This situation is called *supervised learning* and requires a teacher to provide the tags.[†] The winner is determined like before with (2-7) or (2-8). In addition to the update of the weights $\mathbf{w}_i$, the responses $\boldsymbol{r}_i$ are adapted using basically the same rule (2-9) with possibly different learning rates λ' and neighborhood functions ν':

$$\Delta \boldsymbol{r}_i = \nu'(\boldsymbol{p}_w, \boldsymbol{p}_i, t) \cdot \lambda'(t) \cdot (\hat{\boldsymbol{r}} - \boldsymbol{r}_i) \; . \tag{2-14}$$

Depending on the actual external topology and the response space $\boldsymbol{R}$, a different operator to measure the difference $\hat{\boldsymbol{r}} - \boldsymbol{r}_i$ between the given correct response and the response stored in the node may be required. Likewise, the update of the response values must be carried out in accordance with the constraints of the actual response space (this issue will be discussed in more detail in chapter 3, "Pose Representations").

The response stored in the winner becomes the overall network response:

$$\tilde{\boldsymbol{r}} = \boldsymbol{r}_w \; . \tag{2-15}$$

[†] If explicit teaching is difficult or impossible, the algorithm must produce the tags $\hat{\boldsymbol{r}}$ independently (*unsupervised learning*). In the absence of further information this requires a stochastic search in the space $\boldsymbol{R}$ of all possible tags with the help of a suitable evaluation function (see [52] for an example).

2.4. Interpolation Between Nodes

As the name suggests, competitive learning, which was introduced in section 2.2.1 on page 17, is realized by the winner-takes-all principle, which leads to a discretization of the response space $\boldsymbol{R}$. However, other neurons may be close runner-ups to the winner, and their responses can be utilized to improve the accuracy of the overall network output $\tilde{\boldsymbol{r}}$. The winner-takes-all rule (2-15) is replaced by some kind of interpolation between several nodes.

Geometrical and topological interpolation are investigated by Göppert and Rosenstiel in [13] and will be outlined below. An even more sophisticated method using iterative matrix inversion is discussed in [15], but improvements over the other methods are minor and disappear in the presence of noise.

2.4.1. Geometrical Interpolation

This method makes use of geometrical information in the (Euclidean) input space $\boldsymbol{P}$: apart from the overall winner w, n additional winners $w_1, \ldots, w_n$ are selected in the order of the distance of their weight vectors to the input vector, i.e.

$$\left\|\hat{\mathbf{w}} - \mathbf{w}_w\right\| \le \left\|\hat{\mathbf{w}} - \mathbf{w}_{w_1}\right\| \le \ldots \le \left\|\hat{\mathbf{w}} - \mathbf{w}_{w_n}\right\| \le \left\|\hat{\mathbf{w}} - \mathbf{w}_i\right\| \qquad \forall i \neq w_{1\ldots n} \tag{2-16}$$

Now it is attempted to improve an approximation $\tilde{\mathbf{w}}$ of the input vector $\hat{\mathbf{w}}$ iteratively by projecting the remaining error $\hat{\mathbf{w}} - \tilde{\mathbf{w}}$ onto the distance vector from the winners. This is repeated n times for each winner w_j in order. Concurrently, an approximation $\tilde{\boldsymbol{r}}$ of the response is calculated in the same fashion. Starting with the overall winner w to initialize $\tilde{\mathbf{w}}_0 = \mathbf{w}_w$ and $\tilde{\boldsymbol{r}}_0 = \boldsymbol{r}_w$, the normalized scalar product κ_j representing the optimal fraction to go in the direction of the next winner's weight vector is determined thus:

$$\kappa_j = \frac{(\hat{\mathbf{w}} - \tilde{\mathbf{w}}_{j-1}) \cdot (\mathbf{w}_{w_j} - \tilde{\mathbf{w}}_{j-1})}{\left\|\mathbf{w}_{w_j} - \tilde{\mathbf{w}}_{j-1}\right\|^2} \,. \tag{2-17}$$

This fraction is used to update input and output approximations:

$$\tilde{\mathbf{w}}_j = \tilde{\mathbf{w}}_{j-1} + \kappa_j(\mathbf{w}_{w_j} - \tilde{\mathbf{w}}_{j-1}), \tag{2-18}$$

$$\tilde{\boldsymbol{r}}_j = \tilde{\boldsymbol{r}}_{j-1} + \kappa_j(\boldsymbol{r}_{w_j} - \tilde{\boldsymbol{r}}_{j-1}). \tag{2-19}$$

The overall network response $\tilde{\boldsymbol{r}}$ is taken to be $\tilde{\boldsymbol{r}}_n$, the n^{th} approximation.

2.4.2. Topological Interpolation

Topological interpolation takes into account the topology preservation of the self-organizing feature map. Depending on the complexity and nonlinearity of the input-output relation, similar input vectors may produce quite different responses. Since the winners can only be determined based on the similarity in the input space, geometrical interpolation may lead to erroneous results in the approximation of the output. Topological interpolation tries to overcome this problem by selecting the relevant nodes based on the vicinity to the winner in

the response space $\boldsymbol{R}$; interpolation takes place between the winner and its n nearest topological neighbors. For every neighbor j of the winner w, we calculate its contribution κ_j similar to (2-17):

$$\kappa_j = \frac{(\hat{\mathbf{w}} - \mathbf{w}_w) \cdot (\mathbf{w}_j - \mathbf{w}_w)}{\|\mathbf{w}_j - \mathbf{w}_w\|^2} \ . \tag{2-20}$$

The overall network response is the weighted sum of the contributions of the n nearest topological neighbors of the winner in $\boldsymbol{R}$:

$$\tilde{\boldsymbol{r}} = \boldsymbol{r}_w + \frac{1}{n} \sum_{j=1}^{n} \kappa_j (\boldsymbol{r}_j - \boldsymbol{r}_w). \tag{2-21}$$

2.5. Local Linear Maps

The accuracy of the basic approach can be significantly enhanced by making better use of the response values $\boldsymbol{r}_i$. Instead of using them directly as discrete outputs, they can serve as points of support in an additional linear network [39][42], essentially producing a smoother mapping. The entire structure is termed *local linear map* [52]. Each node i additionally stores a matrix $\mathbf{L}_i$. The self-organizing map now acts as a master network that switches between a set of linear slave networks by selecting a winner w, whose matrix $\mathbf{L}_w$ is used to transform the input vector $\hat{\mathbf{w}}$ into the network response

$$\tilde{\mathbf{r}} = \mathbf{r}_w + \mathbf{L}_w(\hat{\mathbf{w}} - \mathbf{w}_w) \ . \tag{2-22}$$

Due to the linear nature of this transformation, (2-22) only makes sense for Euclidean response spaces, where the coefficients of the response values are equivalent and no particular constraints must be satisfied. In other words, the response values must be restricted to vectors, which is denoted by using $\mathbf{r}$ instead of the more general $\boldsymbol{r}$.

The local linear map no longer works as a mere classifier, whose response is limited to selecting one of a set of possible (fixed) responses; instead the overall network response $\tilde{\mathbf{r}}$ becomes a smooth function of the current input, which uses the corresponding $\mathbf{r}_w$ as its point of support. This strategy greatly reduces the number of nodes necessary for obtaining a given accuracy.

Input weights and response values are updated like before using (2-9) and (2-14) respectively. A suitable update rule for the matrices is

$$\Delta\mathbf{L}_i = \nu''(\boldsymbol{p}_w, \boldsymbol{p}_i, t) \cdot \lambda''(t) \cdot (\hat{\mathbf{r}} - \mathbf{r}_i - \mathbf{L}_i(\hat{\mathbf{w}} - \mathbf{w}_i))(\hat{\mathbf{w}} - \mathbf{w}_i)^{\mathrm{T}} \cdot \mathbf{G} \ , \tag{2-23}$$

where $\mathbf{G}$ is a gain matrix whose coefficients may be set to $\|\hat{\mathbf{w}} - \mathbf{w}_i\|^{-2}$ for a start [53]. It turns out, however, that it does not make much of a difference if (2-23) is applied to all nodes or just to the winner:

$$\Delta\mathbf{L}_w = \lambda''(t) \cdot (\hat{\mathbf{r}} - \tilde{\mathbf{r}})(\hat{\mathbf{w}} - \mathbf{w}_w)^{\mathrm{T}} \cdot \mathbf{G} \ . \tag{2-24}$$

This was to be expected because the linear transformation is really independent of the organization of the network and its neighborhood relations and is valid only in the region surrounding the current winner.†

2.6. Rigid Maps

As will be shown in chapter 5, “Results”, self-organization does not always work satisfactorily. In such cases it may just as well be abandoned. In order to do this, the internal and external topology of the network are merged into one, i.e. $\boldsymbol{r}_i \equiv \boldsymbol{p}_i$. This is equivalent to setting the learning rate λ' in (2-14) to zero throughout the training, which is why we will refer to this design as a rigid map (RM). In other words, only the weight vectors of the nodes are adapted during training. The overall network response is equivalent to the position of the winner:

$$\tilde{\boldsymbol{r}} = \boldsymbol{p}_w . \tag{2-25}$$

During training, the winner can no longer be determined according to (2-7) or (2-8) on the basis of the best-matching weight vector, because the weight vectors are still initialized with random values. Instead, supervised training is required, and the winner w is the node whose position in response space most closely matches the parameters $\hat{\boldsymbol{r}}$ used to create the current training view:

$$\delta(\hat{\boldsymbol{r}}, \boldsymbol{r}_w) \le \delta(\hat{\boldsymbol{r}}, \boldsymbol{r}_i) \quad \forall i . \tag{2-26}$$

After training, the winner is of course determined by rule (2-7) or (2-8) again.

This design can only be utilized if the response space is known in advance, which is certainly the case with pose estimation. The best results are obtained when the nodes are positioned in such a way that they evenly quantize the given space spanned by the training views. Compare the map displayed in Figure 2-6 with the ones in Figure 2-2 to see the difference.

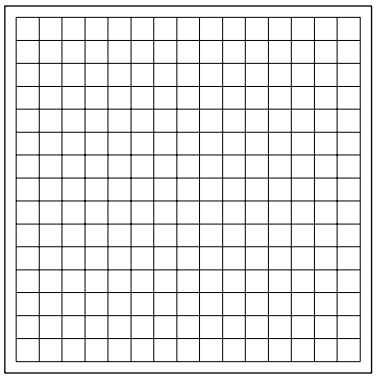


Figure 2-6: 16x16 nodes of a two-dimensional rigid map with a square topology. The nodes remain at their positions throughout the training, and only the weight vectors are updated.

† This may have been Ritter’s intention anyway, but he is not quite clear on that.

2.7. Previous Research

Numerous journal and conference papers deal with object recognition by neural networks, but few specifically address the problem of pose estimation by neural networks, though some exist that try to tackle the problem analytically [5][32][62]. Of the seven neural network approaches discussed below, three avoid the problem of self-occlusion by working with transparent objects, and some heavily depend on knowing the correspondences between image and model features. Others impose relatively narrow range limitations, and except for Miller and Gilmore [43], all work exclusively with computer-generated input data.

Khotanzad and Liou [23] combine an object recognition network with a bank of pose estimation networks (one for every object). The pose estimation modules consist of two stages. The first stage is a quadrant classifier with four outputs (one for each quadrant in the upper hemisphere) that activate the corresponding network in the second stage, which consists of four networks with two real-valued outputs (the aspect and elevation angles). The binarized silhouette of the object and so-called Zernike moments are extracted from computer-generated images and used as input. For pose estimation, the two best-matching nodes are considered, and the decision is counted as a correct one if one of the two is the correct choice. This way, aspect and elevation angles of undisturbed views can be estimated with less than $3°$ error.

Lu, Lo and Don [35] also combine an object recognition network with a bank of pose estimation networks. A set of 11 different three-dimensional moment invariants is generated from complex moments of two-and-a-half-dimensional range images. This data is used as the input vector for both the recognition and the pose estimation networks. The pose parameters are the longitude and latitude on a sphere enclosing the object. The output is claimed correct if the object is identified and the error of the estimated angles is less than $5°$. Tests with sample airplane and machine part models yield diverse results: Depending on the number of training steps and hidden nodes, correct identification and pose estimation rates range from 60 to 95 percent.

Maggioni and Wirtz [37] train a self-organizing feature map of 5^3 nodes with cubical topology. A feature vector consisting of the ordered $[x, y]$ coordinates of the projected object's 18 vertices is used as input, and three angles ranging from 0 to 90 degrees describing the rotation about the object's principal axes are used as response values. 50000 training views of the object are presented to the network. Unfortunately, the results reported are rather cryptic. In practice, this kind of feature vector not only poses the difficult correspondence problem of relating the vertices in the image to the vertices of the object model, but also is not applicable to opaque objects, where self-occlusion comes into play.

Miller and Gilmore [43] work with a set of high-order perspective-invariant spatial relations between lines. The Hopfield-Tank optimization network is used to match model and image line segments. They make the simplifying assumption that vertical lines in the object project onto vertical lines in the image. Real-world images are used as input, but correct matching of correspondences heavily depends on the contrast of the images.

Park and Cannon [46] use a three-layer feed-forward network with 480 nodes to classify views of an object as one of 80 training views, which are generated by positioning the camera at 80 different points on a sphere around the object. After the extraction of the object's boundary lines, the centroid of the silhouette is calculated. Equally spaced sample points of the centroidal profile (i.e. the distance from the centroid to points on the boundaries) of this silhouette are fed into the neural network. The authors report problems especially in the reliable selection of the starting point of an object's centroidal profile, which generally necessitates the generation and verification of several pose hypotheses.

Poggio and Edelman [49], like Maggioni and Wirtz [37], also use the projected $[x, y]$ coordinates of an object's vertices as the input vector to their network, hence their approach suffers from the same limitations. They employ Generalized Radial Basis Functions (GRBFs), a scheme for the approximation of smooth functions, whose weights are adapted during training via matrix inversion. The output of the network is the linear superposition of the activities of all its basis units. After presenting roughly 100 training views to the network, they find the performance to be satisfactory with an RMS error of less than $10°$ for longitude and latitude for viewing regions up to an octant of the sphere, but it degrades to an error of 50 to 60 degrees when the full sphere is used to generate the views.

Ritter [53] employs small self-organizing feature maps with cubical topology to briefly demonstrate learning with his local linear maps (see section 2.5 on page 23 for a more detailed discussion of this concept). He uses a set of 1000 pairs of vertically and horizontally Sobel-filtered images of an opaque computer-generated robot gripper. After subsampling such an image pair to 8x8 pixels, the resulting 2x8x8 values are fed into the network. The responses of the network are three angles describing the rotation about the object's principal axes; all three are limited to a range of 90 degrees. Ritter reports an average error of $5°$ for the smallest network (a cube of 2^3 nodes) and an average error of $3°$ for the largest (a cube of 4^3 nodes) after a total of 20000 training steps.

Let no one ignorant of geometry enter my door.

Plato

3 Pose Representations

The goal of pose estimation is a description of the object's translation and rotation with respect to some coordinate system. Mathematically, pose can be represented in various ways [9]. In computer graphics, homogeneous coordinates with their 4x4 transformation matrices are a powerful and common means of describing general transformations. However, due to their generality they are highly redundant when storing merely rotation information and impose many constraints on their coefficients for the matrix to be valid. Fulfilling these orthonormality constraints presents unnecessary difficulties for our computations. Also, calculations are not carried out with infinite precision. The product of orthonormal matrices may no longer be orthonormal, for example, and it is quite difficult to find the nearest orthonormal matrix. The representations dealt with in this chapter, namely roll, pitch and yaw, extended spherical coordinates and quaternions, are much easier to handle in these regards. The function of the neural network heavily depends on correct measures for the difference of two poses, smooth interpolation between poses and an unambiguous conversion of the parameters to and from homogeneous coordinates. We will attempt to determine the most suitable representation for this purpose.

In most cases found in pertinent literature on neural networks, Kohonen maps with an internal two- or three-dimensional Euclidean topology are employed. This topology may not always be the optimal solution, though. As it turns out in the case of pose estimation, it is most advantageous to shape the neural network in harmony with the topology of the problem at hand – provided this topology is known in advance – even more so if the topology strongly differs from a Euclidean one. Depending on the kind of representation used, the network can now be laid out in three different topologies. The problem to be solved here is a quantization of the respective spaces that is as uniform as possible.

3.1. Roll, Pitch and Yaw

Translation and rotation information about an object can be compressed into a 6-dimensional vector containing roll, pitch and yaw (i.e. the angles of rotation about the three principal axes z, y and x respectively) as well as the translation coefficients along each of the three axes. The corresponding transformation matrix **M** that transforms a point given in one coordinate system into another system is calculated from the vector $[r_x, r_y, r_z; t_x, t_y, t_z]$ describing the relation of the two systems as follows:

$$\mathbf{M} = \begin{bmatrix} \cos r_z & \sin r_z & 0 & 0 \\ -\sin r_z & \cos r_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos r_y & 0 & -\sin r_y & 0 \\ 0 & 1 & 0 & 0 \\ \sin r_y & 0 & \cos r_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos r_x & \sin r_x & 0 \\ 0 & -\sin r_x & \cos r_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (3\text{-}1)$$

An object's pose can be described as the translation and rotation of the object with respect to its own coordinate system. We consider rotation only, so the position information ***p*** becomes a vector in three-dimensional Euclidean space:

$$\mathbf{p} = [r_x, r_y, r_z] . \quad (3\text{-}2)$$

Distance and interpolation calculations between two positions $\mathbf{p}_1$ and $\mathbf{p}_2$ follow the common rules of Euclidean space:

$$\delta(\mathbf{p}_1, \mathbf{p}_2) = \|\mathbf{p}_1 - \mathbf{p}_2\|, \quad (3\text{-}3)$$

$$\mathbf{p}_3(k) = (1-k)\mathbf{p}_1 + k\mathbf{p}_2 , \quad (3\text{-}4)$$

with the interpolated $\mathbf{p}_3$ coming to lie a fraction k of the distance between $\mathbf{p}_1$ and $\mathbf{p}_2$.

There are however two drawbacks of the roll-pitch-yaw representation, which severely limit its usability for our application.

- It is highly ambiguous; a unique pose can be described by different angles. Rotating an object 180° about its x-axis will yield the same pose as rotating it 180° about its y-axis and 180° about its z-axis, for instance.
- It does not allow for natural distance calculation or interpolation. A rotation of 90° about the z-axis and then 90° about the y-axis has the effect of a 120° rotation about the axis $[1, 1, 1]$, but rotating 30° about the z-axis and 30° about the y-axis does not give a rotation of 40° about the axis $[1, 1, 1]$. Instead this results approximately in a rotation of 42° about the axis $[1, 0.3, 1]$.

3.1.1. Cubical Topology

For the roll-pitch-yaw representation a cubical topology can be chosen. The nodes are positioned on a regular cubical lattice in such a way that the given range of rotations in each dimension is quantized as shown in Figure 3-1. Mathematically, the coefficients of the three-dimensional position vector **p** of each node are calculated thus:

$$\mathbf{p}_{x,y,z} = ll + \frac{k-0.5}{\sqrt[3]{N}}(ul-ll)\ ,\ k = 1\ldots\sqrt[3]{N}\ . \tag{3-5}$$

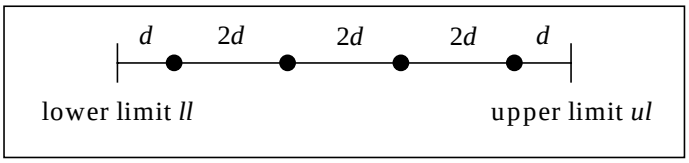


Figure 3-1: Regular distribution of nodes in cubical topology along one dimension.

An entire cubical network consisting of 3^3 nodes is depicted in Figure 3-2.

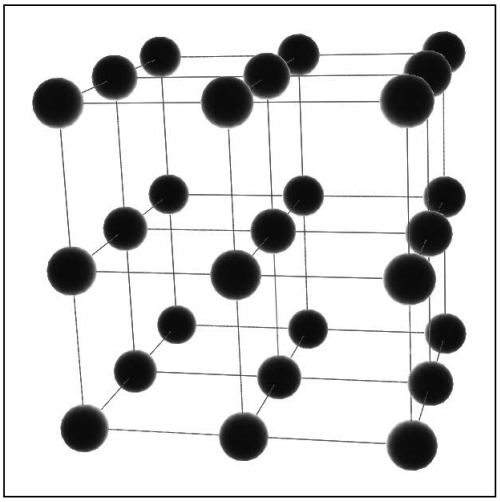


Figure 3-2: Cubical network of 3^3 nodes. The positions of the nodes in space are determined by their **p**, and the lines connect at most six nearest neighbors in ***P***.

3.2. Extended Spherical Coordinates

Roll, pitch and yaw describe the orientation of an object with respect to its own coordinate system. Alternatively, different poses of an object can also be produced by moving the camera around the object on an imaginary sphere with the object in its center. Considering our application and coordinate systems, this provides a very natural representation. Translations can be removed by keeping the camera aimed at the center of the object and keeping the radius of the sphere constant. This method again yields three parameters, two denoting longitude and latitude, and one for the rotation φ of the camera about its own axis, the line of sight (see Figure 3-3). It reduces the manifold ambiguities of the roll-pitch-yaw representation to only two points, namely the poles of the sphere, where the latitude is ±90° and the longitude is undefined.

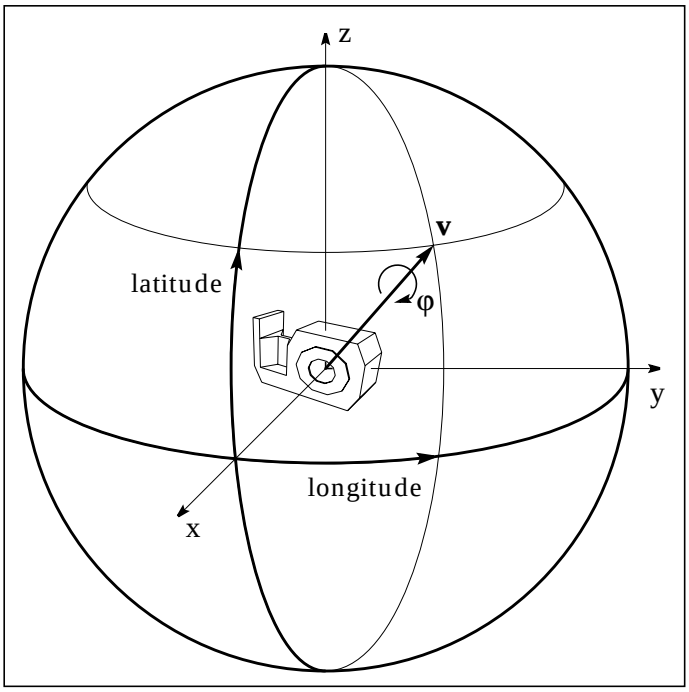


Figure 3-3: Extended spherical coordinates. The camera remains on the surface of the sphere and moves around the object in the sphere's center. The location of the camera is given by latitude and longitude or by the unit vector **v** alternatively. The rotation of the camera about **v** (the line of sight) is determined by φ.

Instead of longitude and latitude we use a Euclidean unit vector **v** describing the position on the sphere from which the object is viewed in this implementation for easier distance and interpolation calculations. Hence, a pose is really represented by this unit vector and an angle, $[\mathbf{v}, \varphi]$.

The distance between two poses $[\mathbf{v}_1, \varphi_1]$ and $[\mathbf{v}_2, \varphi_2]$ comprises two separate parts: the spherical distance

$$\delta_S(\mathbf{v}_1, \mathbf{v}_2) = \operatorname{acos}(\mathbf{v}_1 \cdot \mathbf{v}_2) \tag{3-6}$$

along a great arc between the two positions $\mathbf{v}_1$ and $\mathbf{v}_2$ on the unit sphere, and the circular distance

$$\delta_C(\varphi_1, \varphi_2) = \omega(\varphi_1 - \varphi_2) \tag{3-7}$$

between the two rotations about the line of sight, to which an offset of 2π or -2π must be added if necessary to fit it within the range of $\pm\pi$, indicated by the "wrap-around" function ω. The square sum of both distance measures yields the total distance

$$\delta(\boldsymbol{p}_1, \boldsymbol{p}_2) = \sqrt{\delta_S(\mathbf{v}_1, \mathbf{v}_2)^2 + \delta_C(\varphi_1, \varphi_2)^2}\,. \tag{3-8}$$

Interpolation between two different views or poses $[\mathbf{v}_1, \varphi_1]$ and $[\mathbf{v}_2, \varphi_2]$ also consists of two steps. First interpolation on the unit sphere follows the shortest path between the two positions $\mathbf{v}_1$ and $\mathbf{v}_2$, which is again a great arc. This kind of *s*pherical *l*inear int*erp*olation is often referred to as *slerp* [9]. Suppose $\mathbf{v}_3$ should come to lie a fraction k of the angle δ between $\mathbf{v}_1$ and $\mathbf{v}_2$ ($\cos\delta = \mathbf{v}_1 \cdot \mathbf{v}_2$). From geometry comes

$$\mathbf{v}_3(k) = \frac{\sin(1-k)\delta}{\sin\delta} \cdot \mathbf{v}_1 + \frac{\sin k\delta}{\sin\delta} \cdot \mathbf{v}_2\ , \tag{3-9}$$

which can easily be proven to satisfy the three necessary conditions

$$\begin{aligned} \mathbf{v}_1 \cdot \mathbf{v}_3 &= \cos k\delta, \\ \mathbf{v}_2 \cdot \mathbf{v}_3 &= \cos(1-k)\delta, \\ \mathbf{v}_3 \cdot \mathbf{v}_3 &= 1, \end{aligned} \tag{3-10}$$

using simple trigonometric identities. Notice that (3-9) works only for $\delta \neq 0, \pi$.

Second comes circular linear interpolation between φ_1 and φ_2, which could also be calculated using (3-9), but on a circle this is not really necessary, and since the angles are given,

$$\varphi_3 = \omega(\varphi_1 + k\omega(\varphi_2 - \varphi_1)) \tag{3-11}$$

yields the same results if the special distance and wrap-around properties of the angles are considered again.

Conversion between spherical coordinates and homogeneous coordinates is achieved by creating a coordinate system at the point of view with the z-axis determined by **v**. The x- and y-axes come to lie in the tangential plane through **v**. In order to orient these two axes, it is necessary to choose an absolute reference vector in the object coordinate system against which the rotation φ about the line of sight can be measured. Only this action establishes the two poles of the sphere as the ambiguities in the spherical coordinate representation, because there the reference vector is parallel to the z-axis of the local coordinate system, and the orientation of the x- and y-axis is undefined.

3.2.1. 2-1-Spherical Topology†

The spherical coordinate representation consists of two independent parameter sets: One describes the location of the camera on a sphere around the object and covers the unit 2-sphere. The other describes the angle of rotation about the line of sight and covers the unit 1-sphere. Hence the name 2-1-spherical topology, which also consists of two separate parts.

The third dimension of the neural network is determined by the angle of rotation φ about the line of sight, which is independent of the other parameters. This dimension is of 1-spherical (i.e. circular) topology ranging from -180° to 180° (see Figure 3-4). The quantization of this dimension is trivial. It should be chosen such that the nodes are approximately as far apart as the nodes on the 2-sphere (compare Table 3-1).

In order to spread a certain number of points or nodes evenly on a 2-sphere, we inscribe a regular polyhedron in it. This way, all the polyhedron's vertices come to lie on its circum-sphere. There are five regular convex polyhedra, also called *Platonic solids*, which can be described by *Schläfli symbols* $\{a, b\}$ denoting a regular polyhedron with *a*-gonal faces, *b* of which meet at each vertex [4]: the tetrahedron $\{3, 3\}$, the octahedron $\{3, 4\}$, the cube $\{4, 3\}$, the icosahedron $\{3, 5\}$, and the dodecahedron $\{5, 3\}$. The cube $\{4, 3\}$, for

† Topologists use the dimension number of the locus itself to describe the sphere: a point-pair is a 0-sphere, the circle is a 1-sphere, the surface of the common sphere is a 2-sphere, and so on. Since we place points on the surfaces of spheres, we shall adopt this notation.

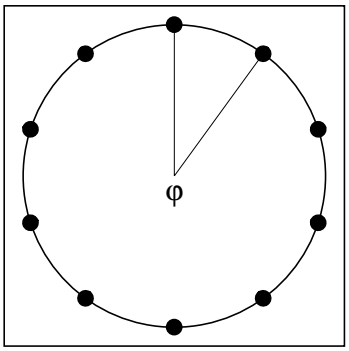


Figure 3-4: Regular distribution of 10 nodes on a 1-sphere (i.e. a circle) determining the angle of rotation φ about the line of sight. The quantization of this dimension should be chosen in harmony with the average distance of nodes on the 2-sphere.

instance, has tetragonal (square) faces, three of which meet at each vertex. Each regular polyhedron $\{a, b\}$ has a reciprocal $\{b, a\}$, which is obtained by replacing each edge by a perpendicular line touching the mid-sphere[†] at the same point.

To make the distribution of points as homogeneous as possible, the icosahedron $\{3, 5\}$ is used; of the five regular polyhedra it is the one with the most faces (20)[‡] and therefore the closest approximation of the sphere (see Figure 3-5a): it encompasses 60.6% of the volume of its circum-sphere.

Recursive subdivision of the triangular faces of the icosahedron into four new triangles is termed *triangulation of the Gaussian Sphere* (see Figure 3-5), which is detailed in [32]. Even though this method naturally cannot produce regular bodies, it leads to a very even distribution of the nodes when they are positioned in the centers of gravity of each of the faces and projected onto the 2-sphere. Another reason for not using the vertices but the centers of the faces instead is that a vertex may have five or six neighbors after subdivision, but every face has three neighbors without exception.

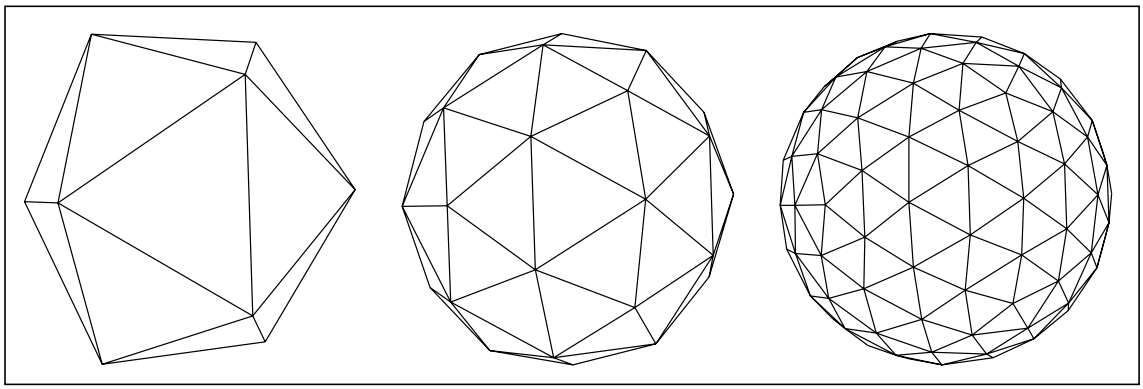


Figure 3-5: The icosahedron and its first two triangulations ($l = 0, 1, 2$). Each triangle is subdivided into four new triangles, whose vertices are then projected onto the 2-sphere.

[†] A polyhedron's mid-sphere touches all edges and contains the in-circles of all its faces.

[‡] A regular polyhedron $\{a, b\}$ has $\frac{4a}{2(a+b)-ab}$ vertices, $\frac{2ab}{2(a+b)-ab}$ edges and $\frac{4b}{2(a+b)-ab}$ faces [4].

The angular distances between neighboring nodes are identical only for the icosahedron at level 0 and begin to vary slightly with rising levels of subdivision. At different levels of subdivision *l*, the number of faces / nodes on the 2-sphere amount to

$$n = 20 \cdot 4^l, \tag{3-12}$$

with $l = 0$ for the original icosahedron (cf. Table 3-1). In order to allow for a greater flexibility with the number of nodes, the centers of the faces at each level of subdivision can be combined with the vertices (at level zero, this is the combination of the icosahedron and the dodecahedron, for example, whose characteristics are given in the column of Table 3-1 that is titled 0+). The regularity of the network is disturbed in that nodes no longer have the same number of neighbors, but this is of minor importance for the problem at hand.

Table 3-1: Network properties

Levels *l* of subdivision	0	0+	1	2
Faces / Nodes on the 2-sphere	20	32	80	320
Minimum angle between neighbors	41.8	37.4	19.9	8.9
Maximum angle between neighbors	41.8	41.8	20.3	10.6
Nodes on the 1-sphere for rotation φ	8	10	16	32
Angle between neighbors	45	36	22.5	11.25
Total number of neighbors per node	5	7 or 8	5	5
Total number of nodes	160	320	1280	10240

An entire 2-1-spherical network consisting of 96 nodes in 0+ arrangement is depicted in Figure 3-6.

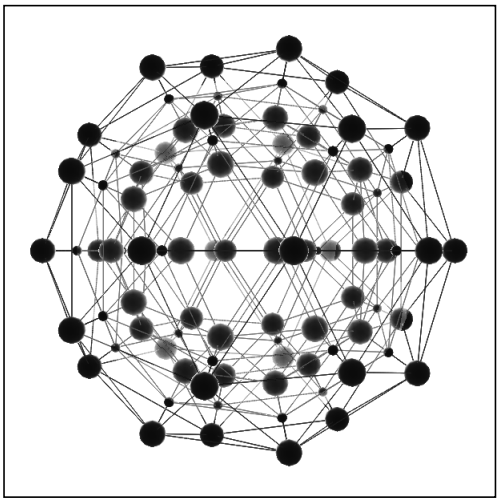


Figure 3-6: 2-1-spherical network with 96 nodes in 0+ arrangement. 32 nodes from the combination of the icosahedron and the dodecahedron are used to quantize the 2-sphere, and only 3 nodes are used for each 1-sphere in order to keep the total number of nodes low. As mentioned in the text and in Table 3-1, the use of 9 or 10 nodes on each 1-sphere is recommended in practice. The size of the nodes is proportional to the absolute value of φ.

3.3. Quaternions

The set of all possible rotations also fits into the coherent algebraic structure of quaternions discovered by Hamilton in 1843 [16]. Quaternions, usually written as $a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}$, constitute a four-dimensional vector space with a basis of four special "unit vectors" 1, $\boldsymbol{i}$, $\boldsymbol{j}$, $\boldsymbol{k}$, which are governed by the fundamental rule

$$\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1 .^{\dagger} \tag{3-13}$$

From (3-13) the other multiplication rules can be derived:

$$\begin{aligned} \boldsymbol{ij} &= \boldsymbol{k} = -\boldsymbol{ji}, \\ \boldsymbol{jk} &= \boldsymbol{i} = -\boldsymbol{kj}, \\ \boldsymbol{ki} &= \boldsymbol{j} = -\boldsymbol{ik}. \end{aligned} \tag{3-14}$$

These rules determine the product of any two quaternions. Notice that multiplication is non-commutative – quaternions form a division ring.[‡]

Multiplication or division results tend to get quite lengthy with the above notation. There exists a much more compact notation for quaternions that dramatically simplifies formulas [56]. In this interpretation, a quaternion $\boldsymbol{q} = a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}$ is represented by the combination of the scalar a and the vector $\mathbf{v} = [b, c, d]$: $\boldsymbol{q} = [a, \mathbf{v}]$. Now the addition formula for two quaternions $\boldsymbol{q}_1 = [a_1, \mathbf{v}_1]$ and $\boldsymbol{q}_2 = [a_2, \mathbf{v}_2]$ becomes

$$\boldsymbol{q}_1 + \boldsymbol{q}_2 = [a_1 + a_2, \mathbf{v}_1 + \mathbf{v}_2] , \tag{3-15}$$

and multiplication is given by

$$\boldsymbol{q}_1\boldsymbol{q}_2 = [a_1a_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, a_1\mathbf{v}_2 + a_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2] . \tag{3-16}$$

The scalar product of two quaternions is defined as

$$\boldsymbol{q}_1 \cdot \boldsymbol{q}_2 = a_1a_2 + \mathbf{v}_1 \cdot \mathbf{v}_2 , \tag{3-17}$$

from which follows the definition of the length of a quaternion

$$\|\boldsymbol{q}\| = \sqrt{a^2 + \|\mathbf{v}\|^2} . \tag{3-18}$$

Taking the conjugate of a quaternion negates its imaginary parts:

$$\boldsymbol{q}^* = a - b\boldsymbol{i} - c\boldsymbol{j} - d\boldsymbol{k} = [a, -\mathbf{v}] . \tag{3-19}$$

Hence, the product of a quaternion with its conjugate is real:

$$\boldsymbol{q}\boldsymbol{q}^* = \boldsymbol{q} \cdot \boldsymbol{q} = \|q\|^2 , \tag{3-20}$$

[†] The quaternions with $c \equiv d \equiv 0$ constitute a subsystem isomorphic with the field of complex numbers.

[‡] A division ring or skew field is a system of elements closed under addition and multiplication, such that under addition the system is a commutative group with identity 0, under multiplication the elements other than 0 form a group, and both distributive laws hold.

from which follows the inverse of a non-zero quaternion:

$$\boldsymbol{q}^{-1} = \frac{\boldsymbol{q}^*}{\boldsymbol{q} \cdot \boldsymbol{q}}. \tag{3-21}$$

In the case of the unit quaternion with $\|\boldsymbol{q}\| = 1$, the inverse is just the conjugate.

It turns out that all rotations correspond to unit quaternions. Rotations and quaternions both form a non-commutative group under multiplication, and in fact these two groups are closely related. Performing successive rotations corresponds to multiplying quaternions according to (3-16). It is interesting to note here that this takes fewer arithmetic operations than multiplying two rotation matrices. A rotation by the angle φ about the unit vector $\mathbf{u}$ is described by the quaternion

$$\boldsymbol{q} = \left[\cos\frac{\varphi}{2}, \mathbf{u}\sin\frac{\varphi}{2}\right]. \tag{3-22}$$

The calculations of interest to us are even more straightforward than they were in the spherical coordinate system of the previous section, because the three parameters are no longer contained in two separate sets. Yet calculations remain quite similar, because unit quaternions also come to lie on a sphere, namely the 3-sphere in 4-space. The only difficulty relevant for our application that remains with the quaternion representation is an ambiguity between a rotation about the axis $\mathbf{u}$ by the angle φ and a rotation about $-\mathbf{u}$ by $-\varphi$; the corresponding quaternions are antipodes on the 3-sphere. This problem can be circumvented by restricting all quaternions to one hemisphere, e.g. the upper one with $a \geq 0$. Furthermore, we need to close this 3-hemisphere at the 2-sphere $a = 0$, i.e. the distance between antipodes on this 2-sphere is assumed to be 0. Consider the two-dimensional analogy (or projection, if you will) in Figure 3-7.

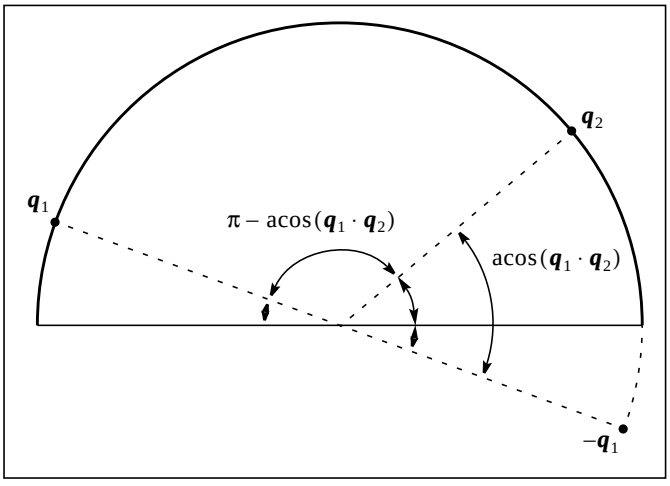


Figure 3-7: Closing a 1-hemisphere with a 0-sphere. The distance between antipodes is assumed to be 0 for distance calculations and interpolation, which always follow a great arc.

The distance between two quaternions $\delta(\boldsymbol{q}_1, \boldsymbol{q}_2)$ is measured as the angle $\operatorname{acos}(\boldsymbol{q}_1 \cdot \boldsymbol{q}_2)$ or $\operatorname{acos}(-\boldsymbol{q}_1 \cdot \boldsymbol{q}_2) = \pi - \operatorname{acos}(\boldsymbol{q}_1 \cdot \boldsymbol{q}_2)$, whichever is smaller (the maximum distance being $\pi/2$).

Interpolation between two quaternions $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ follows the shortest path between them or between $-\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ on the unit 3-sphere, whichever is shorter. The path is the equivalent

of a great arc in 4-space, and in fact spherical linear interpolation (3-9) applies to quaternions as well. If the result comes to lie in the "wrong" hemisphere, its antipode is taken instead.

Conversion between quaternions and homogeneous coordinates is not problematic at all. However, it is quite lengthy, which is why the reader is referred to [21][56]. Again care must be taken that quaternions converted from homogeneous coordinates come to lie in the upper hemisphere.

3.3.1. 3-Hemispherical Topology

For the quaternion representation, the nodes must be distributed evenly on the unit 3-sphere in 4-space. Since good solutions to the related problem in the three-dimensional case were obtained from regular polyhedra, we will study their four-dimensional equivalents; the word *polytope* is the general term in n-dimensional space. Any $n+1$ points which do not lie in an $(n-1)$-space are the vertices of an n-dimensional simplex, the simplest polytope. In 0-space this is a point, in 1-space a line segment, in 2-space a triangle, and in 3-space a tetrahedron. Joining the vertices of a tetrahedron with a fifth point outside its 3-space creates a four-dimensional simplex, the pentatope. When its vertices are mutually equidistant, the simplex is said to be regular. Just as the regular tetrahedron in 3-space is bounded by four equilateral triangles, the five bounding cells of the regular simplex are regular tetrahedra, which are obtained by removing one of the five vertices apiece. The coordinates of one regular simplex can be taken as $[1, 0, 0, 0]$, $[0, 1, 0, 0]$, $[0, 0, 1, 0]$, $[0, 0, 0, 1]$ and $[s, s, s, s]$ with $s = \frac{1}{4}\sqrt{4\sqrt{2}-3} \approx 0.4$.

The Schläfli symbols introduced in section 3.2.1 on page 31 can also be generalized to n-dimensional space. Apart from the regular simplex $\{3, 3, 3\}$ there exist five other regular polytopes in 4-space: the cross-polytope $\{3, 3, 4\}$ with 8 vertices and 16 tetrahedral cells, its reciprocal, the hypercube $\{4, 3, 3\}$, with 16 vertices and 8 cubical cells, the $\{3, 4, 3\}$ with 24 vertices and 24 octahedral cells, as well as the $\{3, 3, 5\}$ with 120 vertices and 600 tetrahedral cells, and its reciprocal $\{5, 3, 3\}$ with 600 vertices and 120 dodecahedral cells.[†] The last two appear to be the most promising for distributing larger numbers of points on the 3-sphere. Brou [2] establishes an approximate measure for the quality of the distribution as the product of each cell's hypersurface area and the number of vertices. The 120 vertices of $\{3, 3, 5\}$ and the 720 vertices of the combination of $\{3, 3, 5\}$ and its reciprocal $\{5, 3, 3\}$ lead to good distributions. Even more points could be obtained by subdividing the regular polytopes with an algorithm similar to the triangulation of the Gaussian sphere in section 3.2.1 on page 31. Because antipodes represent equivalent orientations, only the points in the upper hemisphere with $a \geq 0$ are considered, the other half is thrown away. Vertices on the 2-sphere $a = 0$ that happen to have antipodes are only considered once. These restrictions lead to the networks summarized in Table 3-2. The nodes are positioned in the vertices (v) of the $\{3, 3, 5\}$ as well as in the centers of its cells (c) and faces (f), projected onto the 3-sphere. The v-arrangement corresponds to the $\{3, 3, 5\}$, the c-arrangement to the $\{5, 3, 3\}$, and the vc-arrangement to the above-mentioned combination of the two.

[†] In spaces of more than four dimensions, the only regular polytopes are the regular simplex $\{3, 3, \ldots, 3\}$, the cross-polytope $\{3, \ldots, 3, 4\}$ and the hypercube $\{4, 3, \ldots, 3\}$.

Table 3-2: Network properties

	v	c	vc	f
Number of nodes	60	300	360	600
Minimum angle between neighbors	36.0	15.5	15.5	12.7
Maximum angle between neighbors	36.0	15.5	25.2	20.6
Number of neighbors	12	4	20	12

Unfortunately, means and imagination fail us when it comes to visualization of these arrangements; points on a 3-sphere can hardly be visualized in 3D, much less in 2D print. Figure 3-8 still tries to do just this: The vector part **v** of the unit quaternion is used to specify the position of a node in 3-space, and the scalar part *a* determines its size. We leave out the lines connecting neighbors for reasons of clarity – each one of the 360 nodes in this network has 20 neighbors! If this illustration shows anything, it is the regularity of the network that stems from arranging the nodes in a regular polytope structure.

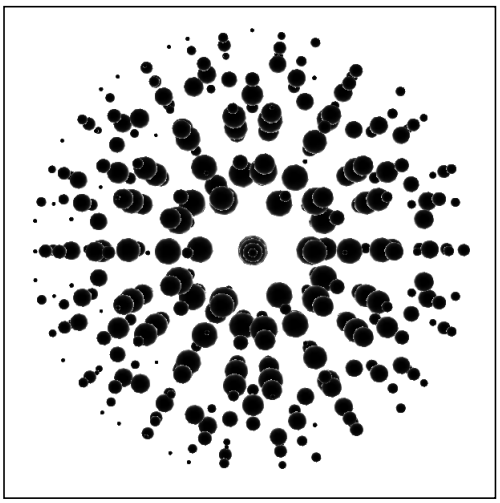


Figure 3-8: 3-hemispherical network with 360 nodes in vc-arrangement. The position of a node in 3-space is given by the vector part **v** of the unit quaternion specifying its ***p***, and its size is determined by the scalar part *a* of the unit quaternion.

3.4. Training Views

During training, random views of the object must be presented to the neural net. Random angles for roll, pitch and yaw can be obtained easily using a random number generator with a uniform distribution. However, due to the ambiguities of this representation, the uniformity of the distribution of the corresponding orientations is questionable.

Distributing points uniformly on a 1-sphere is trivial. However, a “uniform” random distribution of points on the 2-sphere and the 3-sphere is required for spherical coordinates and quaternions. One possible algorithm to generate the desired distribution is the following, which generalizes to any number of dimensions: Each coefficient of a vector is chosen uniformly from the interval $[-1, 1]$. Vectors whose length is greater than one are rejected until a vector within the unit sphere is obtained, which is then normalized to unit length.

On two occasions I have been asked, "Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?"

Charles Babbage

4 Object Features

An important premise of our neural network approach to pose estimation is the possibility of training the network with simulated input images in order to avoid overly long training times with the robot and still be able to use the network on real-world input images. Choosing which object features to simulate and how to preprocess them to obtain the input for the neural network are therefore very important steps. We are going to focus on the objects's edges as features, taking a look at how to simulate and to extract them as well as their orientation.

4.1. Camera Images

Figure 4-1 shows the original camera image of the object that was used for most of the experiments, a tape dispenser. The camera is mounted directly on the robot gripper. Due to the high distortion of the camera, which becomes evident particularly at the outer edges of the object that appear curved rather than straight, camera calibration is imperative. In the calibration process a (nonlinear) hand-eye transformation is computed, which rectifies the camera projection [61]. The linearized camera image can be calculated from the original in real time on dedicated image processing hardware.

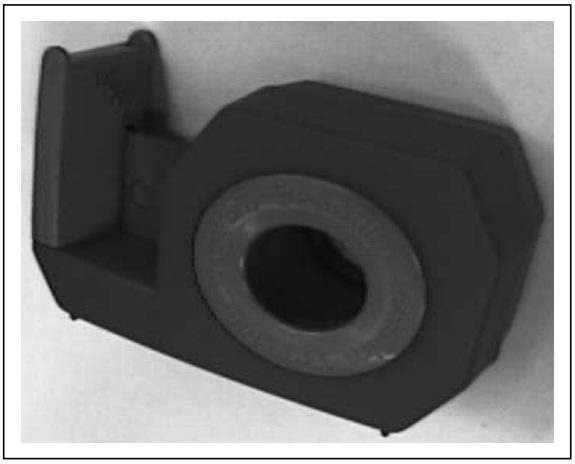


Figure 4-1: Tape dispenser as recorded with the camera on the robot.

Color segmentation is used to separate the object from the background and other objects in the environment by determining the coordinates of a close rectangular bounding box. This segmentation also provides the information about the object's translation in the plane perpendicular to the line of sight. The object's distance from the camera can be determined through inverse perspective projection.

4.2. Features

We decide to use the edge features of the object., because edges are relatively stable features even under variable illumination. Besides, this information can be easily obtained from real-world input images by convolving the camera image with some edge filtering kernel, which can be done in real time on the Datacube (see Figure 4-2).

We intend to use simulated input images for training, because otherwise creating thousands of input images of all possible orientations of an object tagged with the corresponding pose parameters would be a formidable task. In the simulation it is necessary to create a perspective projection of the object onto the camera plane. This is achieved using an analytical hidden line algorithm, which calculates the perspective projection of the object's edges and tags each line either visible, partially visible or not visible [47]; the result is shown in Figure 4-3. Segmentation of the simulated image boils down to finding the bounding rectangle from the outermost vertices of the projected hidden line object.

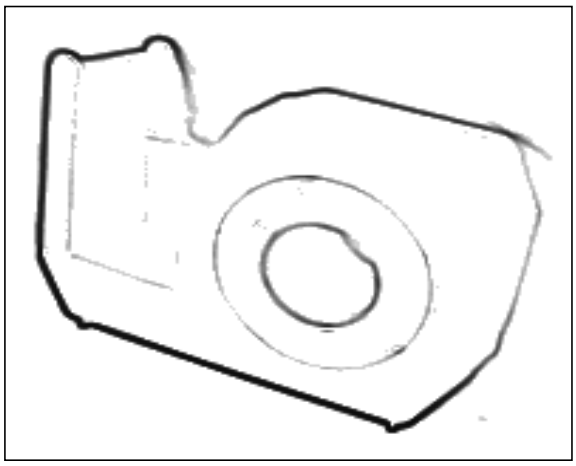


Figure 4-2: Output of edge filter

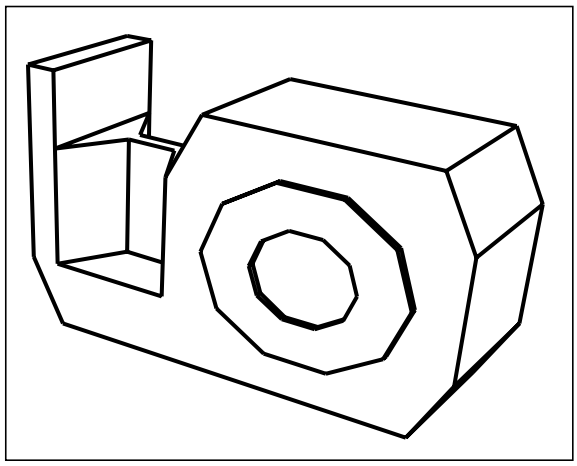


Figure 4-3: Hidden line model

The rectangular area after segmentation usually still contains several ten thousand pixels. In order to reduce the dimension of the input space for the neural net, it is subsampled to a much lower resolution (say 8x8) by subdividing the area into disjoint rectangular blocks and averaging over the pixel values in every one of them (see Figure 4-4). Scanning this subsampled image from left to right and from bottom to top yields the input vector $\hat{\mathbf{w}}$, which is then normalized so as to minimize the difference between simulated and real-world input images and to alleviate the effects of varying illumination.

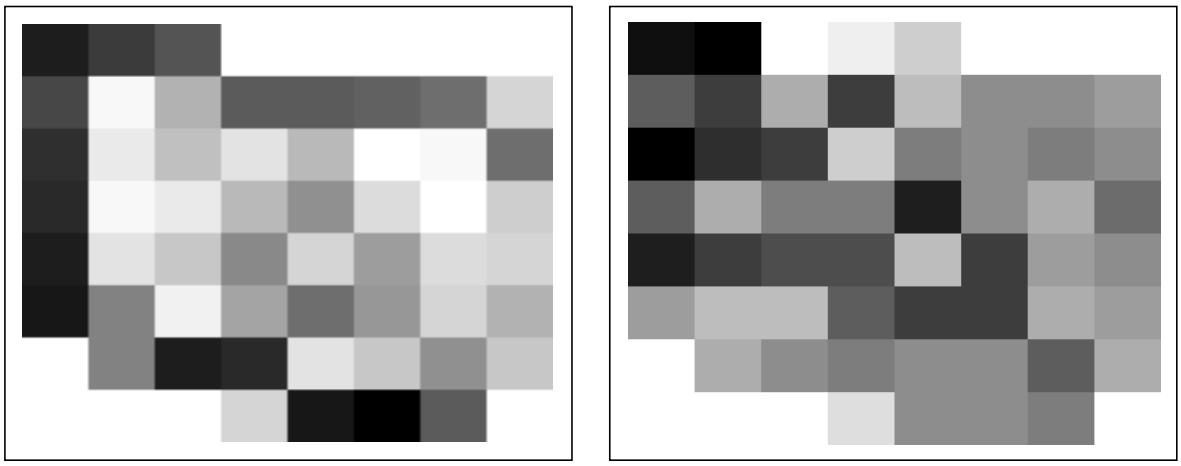


Figure 4-4: Same views of tape dispenser after subsampling to 8x8 pixels

Due to the segmentation, the number of pixels used for calculating one pixel of the subsampled image will generally vary not only with the distance of the object, but also depending on the current viewing position, because the size of the subsampled image is fixed, whereas the size of the segmented image is not.

4.3. Oriented-Edge Filtering

The rigorous subsampling of the hidden line image introduced above inevitably leads to a tremendous loss of information. In particular, the direction of edges that come to lie within a single subsampling block is lost almost completely. Extracting the direction information before subsampling may be expected to alleviate this problem. In fact, appropriately designed two-dimensional wavelets [24][38] or Gabor elementary functions of a suitable modulation frequency [6][41] can serve as oriented-edge filters and differentiate several spatial orientations by virtue of their oriented bandpass nature.

However, the same effect can also be achieved using the gradient of the image. The gradient at each pixel is a two-dimensional vector **v**. While the typical edge filter would disregard the angle $\arg\mathbf{v}$ and choose the pixel values proportional to the length of the gradient, the oriented-edge filter calculates the pixel value p according to

$$p = \|\mathbf{v}\| \cdot f(\arg\mathbf{v} - \varphi). \tag{4-1}$$

The function f is chosen so as to make the filter selective with respect to certain orientations φ. Of course, attention has to be paid to the special properties of its argument, which requires an even function with a periodicity of 180 degrees – a gradient angle offset of 60°, for example, must produce the same filter output as a gradient angle offset of -60°, 120° or 240°. We use a periodically continued Gaussian with spread σ centered at the orientation of interest φ depicted in Figure 4-5.

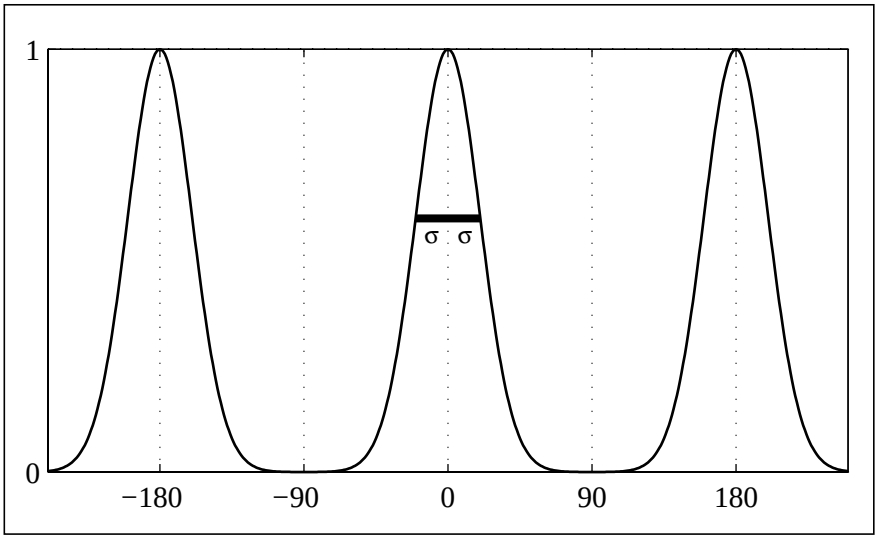


Figure 4-5: Periodic Gaussian used for oriented-edge filtering

In the simulation oriented-edge filtering is particularly simple: The hidden line algorithm provides the starting points and end points of every visible line of the object. From this information, the “gradient” is calculated, which is perpendicular to the angle of a line, and then the same function as above is applied, yielding the intensity of every line.

Figure 4-6 demonstrates the result of oriented-edge filtering the tape dispenser with a Gaussian centered at $\varphi = 0°$, $\varphi = 45°$, $\varphi = 90°$ and $\varphi = 135°$ from the vertical (from top to bottom) and a spread of $\sigma = 20°$. The oriented-edge filtered image is also subsampled like the original hidden line image before. Scanning each image from left to right and from top to bottom again yields a number of vectors (four in this case). They are concatenated to form one input vector.

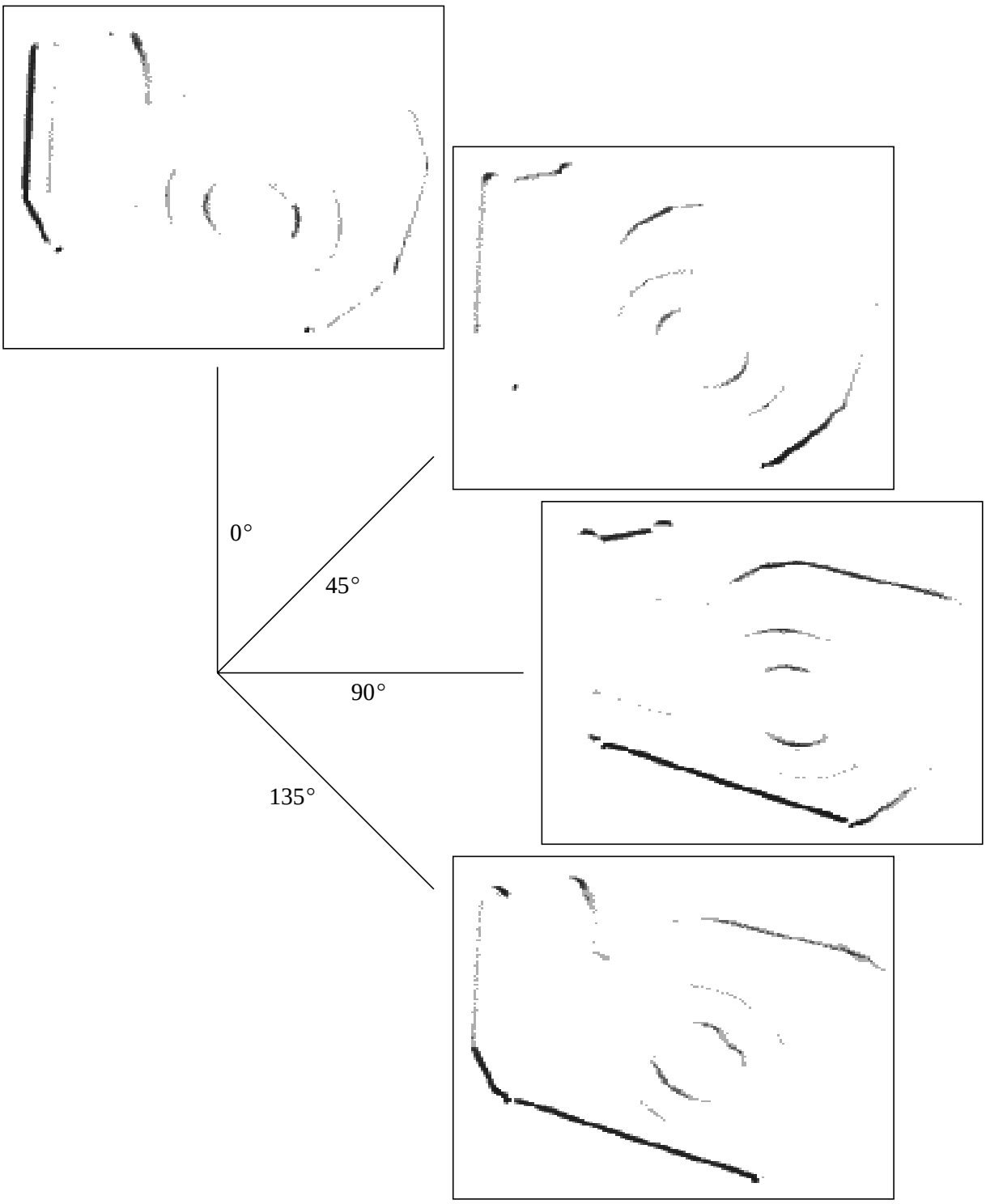


Figure 4-6: Tape dispenser after oriented-edge filtering of four directions

Results! Why man, I have gotten a lot of results.
I know several thousand things that won't work.

Thomas Alva Edison

5 Results

We carried out a multitude of experiments so as to find viable solutions to the problem of automatic pose estimation. On the following pages we examine the performance of the three mathematical representations of pose introduced in chapter 3, "Pose Representations", together with the corresponding network topologies and compare their advantages and disadvantages. We also attempt to determine the effects of possible modifications and enhancements to Kohonen's original self-organizing map design like the local linear map, the rigid map, and interpolation techniques. Furthermore, the influence of certain parameters and external conditions such as noise is investigated, and the results obtained with real world input images are described. Finally, we discuss the computational requirements and the performance of the algorithm.

5.1. Cubical Topology

Encouraged by the achievements of Ritter [53], we begin with a map of internal and external cubical topology using the roll-pitch-yaw representation for the response values. Its 2^3 nodes are trained with 20000 views covering a range of 90 degrees of rotation about each of the three principal axes. Kohonen [27] gives a rule of thumb for determining the number of necessary training steps: they should be at least 500 times the number of nodes. In fact, Figure 5-1 indicates that the network responses have safely converged after 20000 steps – fewer than 10000 training steps may suffice for networks of this size, but 20000 steps should put us on the safe side. The input images are subsampled to a resolution of 8x8 pixels throughout the following experiments unless specified otherwise. The learning rate λ from (2-11) and the spread σ of the neighborhood function ν from (2-12) are both decreased from 1 at the start to 0.01 at the end of the training.

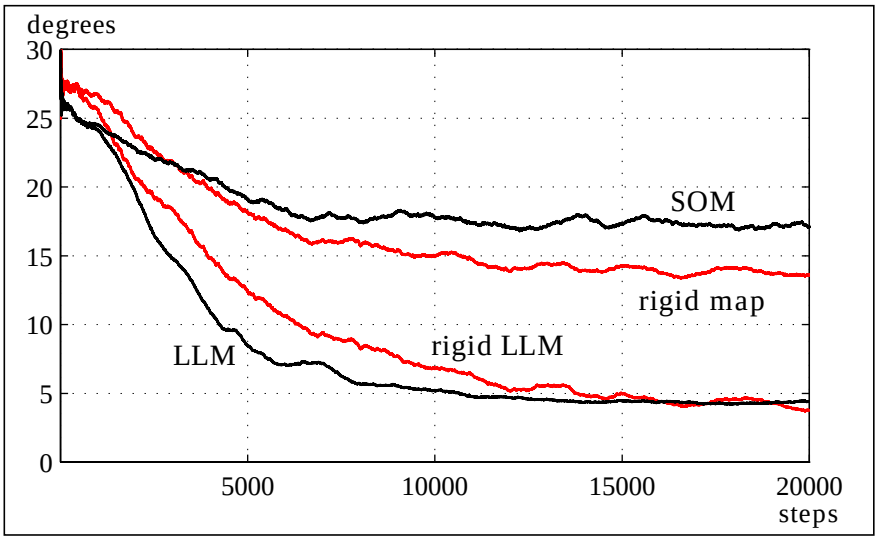


Figure 5-1: RMS error decrease of cubical maps with 2^3 nodes during training. All maps have safely converged after 20000 training steps. However, the self-organizing maps generally converge faster than their rigid counterparts.

The root mean square error ε_{RMS} of the overall network response is used for evaluating the performance of the maps; it is calculated over the three angle differences Δr_x, Δr_y and Δr_z according to

$$\varepsilon_{RMS} = \sqrt{\frac{\Delta r_x^{\,2} + \Delta r_y^{\,2} + \Delta r_z^{\,2}}{3}}. \qquad (5\text{-}1)$$

The error distribution plots below are interpolated histograms that show an approximation of the cumulative distribution function as well as its (re-scaled) derivative, the probability density function, of the RMS error over the range of rotations. They are calculated after training from the network responses to 10000 test views randomly created within the given range of rotations.

5.1.1. Self-Organizing Maps

Figure 5-2 shows the error distribution of the self-organizing map introduced in section 2.2.2 on page 17. Compared to the other methods discussed below, it exhibits by far the worst performance. Not only does it have the highest error rates, but the improvement of the error during training is also very modest, which indicates a lack of self-organization.

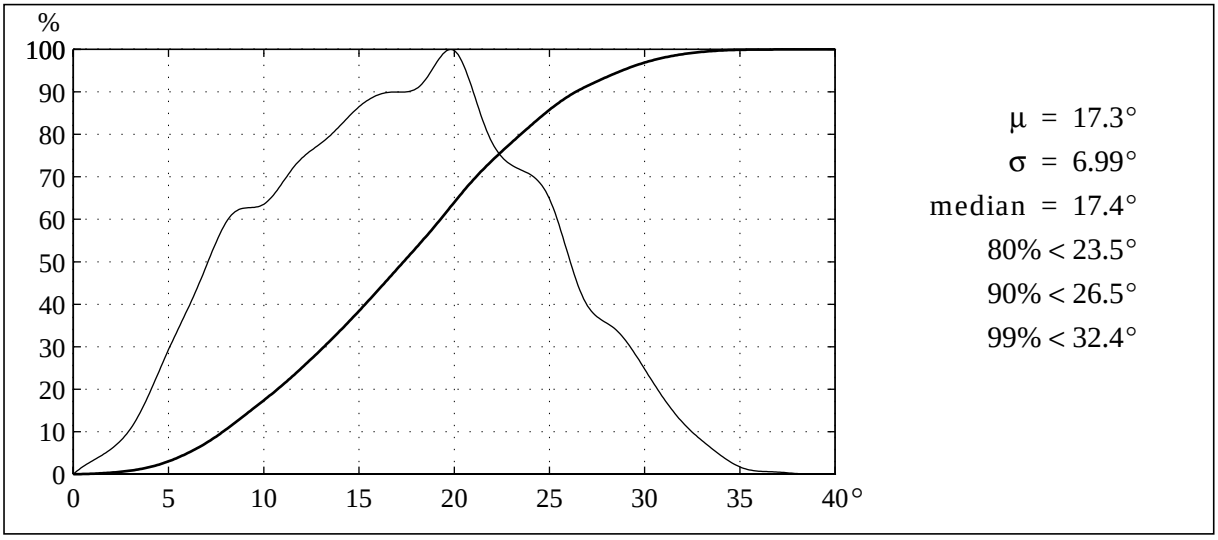


Figure 5-2: RMS error distribution of 90° self-organizing map

Indeed it turns out that the self-organization of the map is utterly dissatisfying except for very small ranges of rotation. Take a look at Figure 5-3, which shows three networks with 3^3 nodes; the positions of the nodes in space are determined by their response vector $\mathbf{r}_i$, and the lines connect nodes that are neighbors in ***P***. The nodes of the left one form an ideal cubical lattice, equidistantly quantizing the given rotation space. The middle one was trained with views of the tape dispenser covering a range of 10 degrees of rotation about each of the three principal

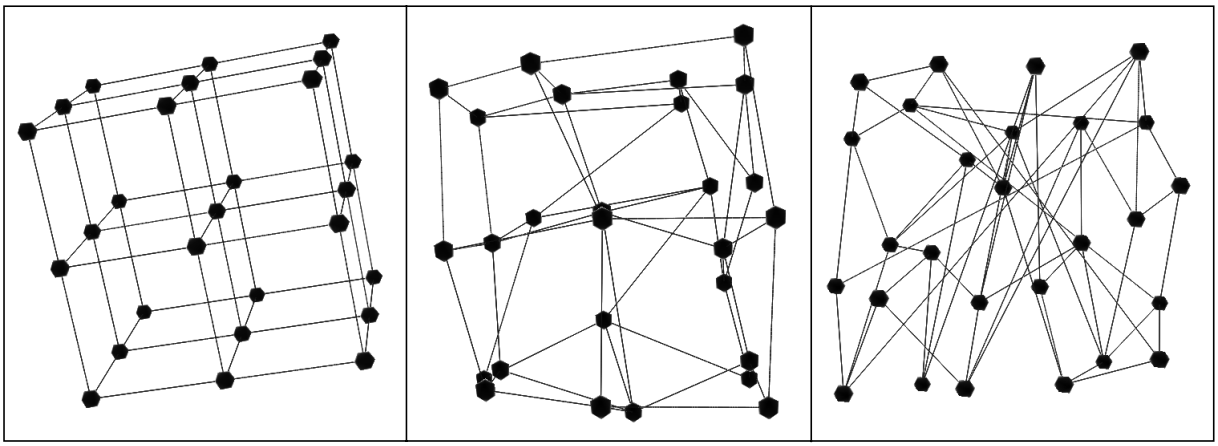


Figure 5-3: Ideal node arrangement (left) and arrangements of self-organizing maps after training with views covering a range of 10 (middle) and 30 (right) degrees. Even for networks trained with views from a very narrow range of rotations, self-organization fails to preserve neighborhood relations.

axes; even though some nodes have moved away considerably from their positions in the cubical lattice, the neighborhood relations of the internal topology are largely preserved. The right one was trained with views covering a range of 30 degrees of rotation about each axis; here the underlying cubical lattice structure is already lost completely. Self-organizing maps trained with views covering a wider range become equally tangled.

This behavior is due to the high discontinuity of the input, which is an object's hidden line image. Both the appearance and the disappearance of edges take place abruptly even when the pose changes smoothly. Therefore the input remains continuous only in small disjoint regions of rotation space, where rotating the object neither produces additional visible edges nor leads to the occlusion of previously visible ones. It is only within these regions that updating the neighbors of the winning node according to (2-9) makes sense. In other words, when the neighbors of the winner lie in a different region of rotation space, which is almost certainly the case for small networks with few nodes, the concurrent and proportional update of their weights can hardly lead to an improvement of the overall response of the map. In fact, the update of the weights can be restricted to the winner alone without a significant decrease in accuracy, particularly when the network is small. This can be achieved by setting the spread σ of the neighborhood function ν from (2-10) to zero throughout the training. Care must be taken however that the initial distribution of weight vectors in input space is such that every node becomes the winner at some point during the early stages of training, because otherwise some nodes may not be trained at all.

5.1.2. Rigid Maps

Since self-organization does not produce the desired results, it may just as well be abandoned. The use of rigid maps as introduced in section 2.6 on page 24 without separation of internal and external topologies together with the new update rule (2-26) can improve the accuracy of the estimations. Figure 5-4 shows the resulting error distribution, which displays improvements of several degrees over the self-organizing map.

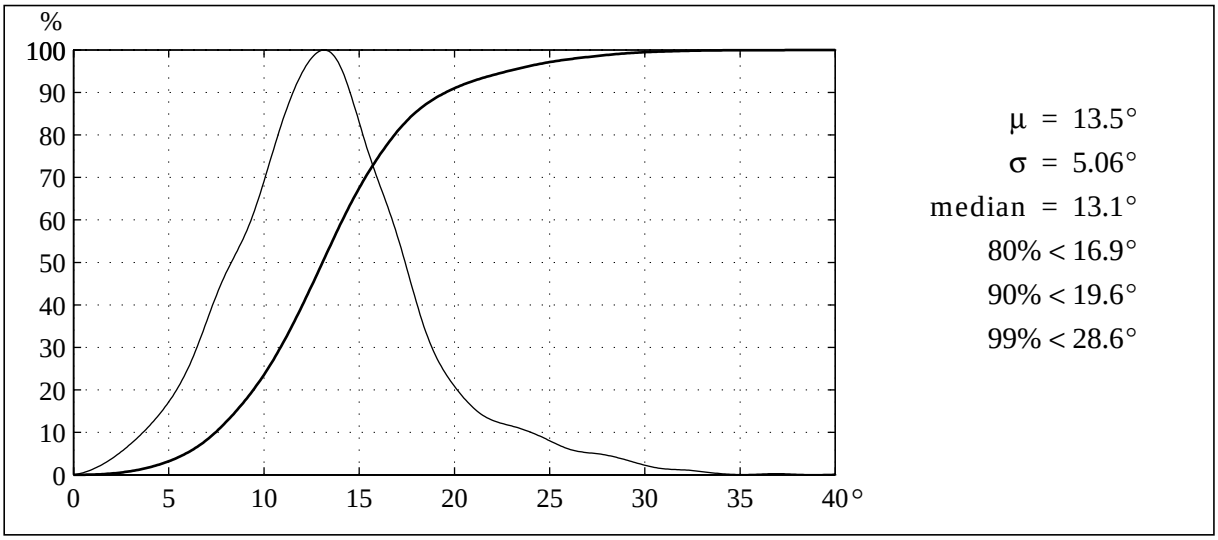


Figure 5-4: RMS error distribution of 90° rigid map

5.1.3. Using More Nodes

Obviously, increasing the number of nodes is the brute-force approach to reducing the absolute error (see Figure 5-5). The local linear map (see the following section) still significantly improves the accuracy for all sizes of networks. However, using more nodes requires more training steps – for the figures given in the plot, the number of training steps was chosen to be 1000 times the number of nodes for each network. Therefore the total training time rises quadratically with the number of nodes in the neural network. Even though this quadratic relationship does not strictly hold for small networks, where most of the training time is spent generating the input vectors (cf. section 5.5 on page 62), it is generally true for networks exceeding a certain size.

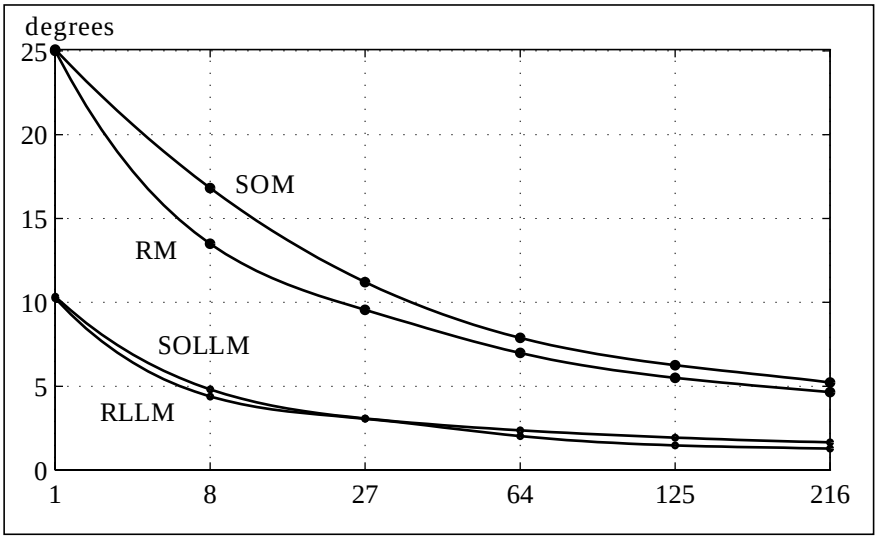


Figure 5-5: Average RMS error in degrees over the number of nodes

5.1.4. Local Linear Maps

The lack of self-organization can be counterbalanced by the use of a local linear map (cf. section 2.5 on page 23). The local linear map has no influence on the primary self-organization of the nodes in the response space, but apparently the nodes arrange in such a way that every node (together with its pertinent LLM-matrix $\mathbf{L}_i$) covers one region of more or less continuous input vectors. Within these regions, an excellent accuracy can be achieved using the local linear transformation (2-22) on the input vector. The reason for this dramatic improvement in accuracy is that the local linear map no longer works as a mere classifier, whose response is limited to selecting one of a set of possible (fixed) responses $\mathbf{r}_i$ stored in the nodes. Instead the overall network response $\tilde{\mathbf{r}}$ becomes a smooth function of the current input using the corresponding $\mathbf{r}_w$ as its point of support.

Figure 5-1 and Figure 5-6 demonstrate the improvements achieved with the linear local map, whose error distribution parameters are three to five times lower than the self-organizing map's. The concept of the local linear map can also be applied to the rigid map, where it produces roughly the same effects (see Figure 5-7).

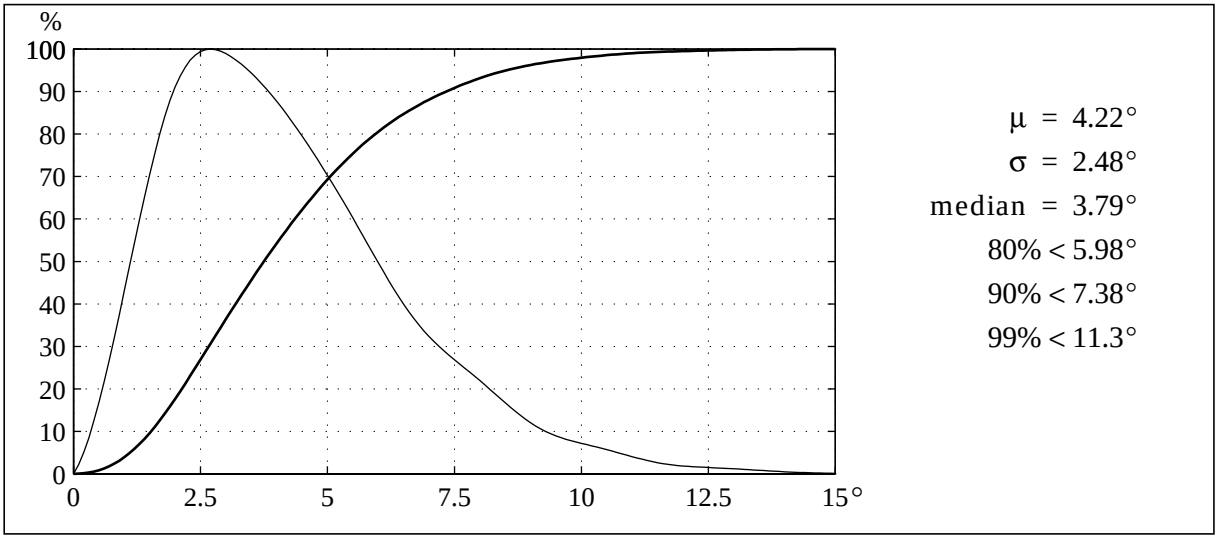


Figure 5-6: RMS error distribution of 90° self-organizing LLM. All distribution parameters are three to five times lower than the self-organizing map's (cf. Figure 5-2).

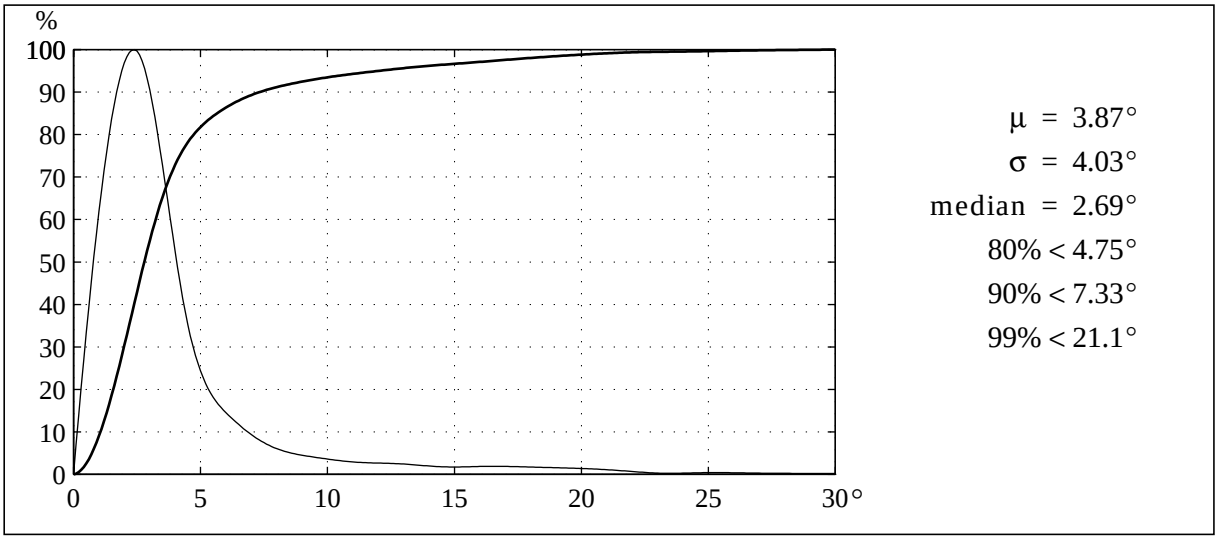


Figure 5-7: RMS error distribution of 90° rigid LLM. All distribution parameters are three to five times lower than the rigid map's (cf. Figure 5-4). However, its performance is slightly inferior to the self-organizing LLM's (cf. Figure 5-6).

5.1.5. Extending the Range

A range of 90 degrees of rotations is nice for demonstration purposes, but it is only of fairly limited practical use. Unfortunately, extending the range to cover all possible orientations turns out to be the crucial hurdle for automatic pose estimation. Figure 5-8 illustrates the problems that occur with wider ranges. The RMS error as a percentage of the range of rotations remains virtually constant for all types of networks up to ranges of nearly 150 degrees. At ranges of rotations above 150 degrees the errors start to soar, and at more than 180 degrees redundancy comes into play, making the established error measures of the roll-pitch-

yaw representation meaningless. Besides, the self-organizing map refuses to spread over the entire range and remains tangled in a small region around the center of the cube.

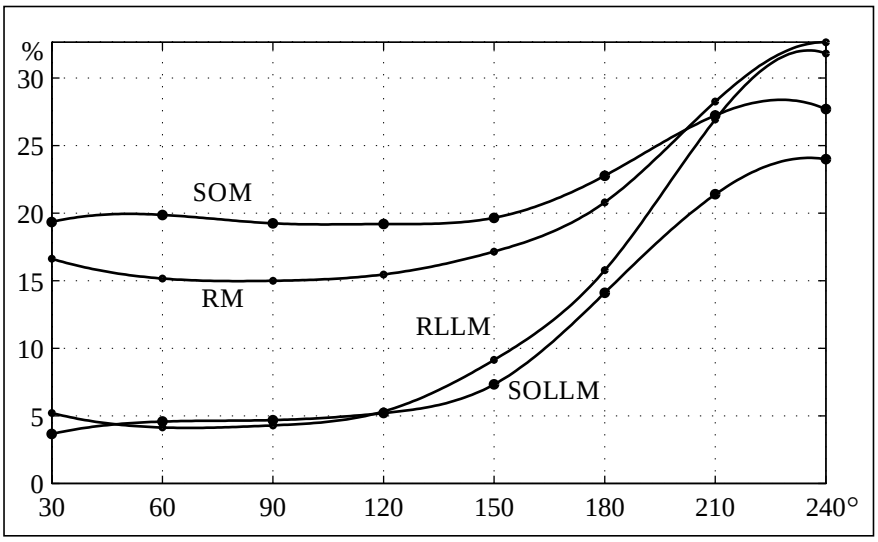


Figure 5-8: The RMS error as a percentage of the range of rotations remains virtually constant for all types of networks up to ranges of nearly 150 degrees. Above this range, the error measures of the roll-pitch-yaw representation become meaningless.

These results inspire the usage of more than one network for the entire 360-degree cube of possible angles. In an effort to keep the total number of nodes as low as possible, local linear maps with 2^3 nodes seem to be the best choice. The LLM covering a range of rotations of 120 degrees is not only the upper boundary of the quasi-linear region in Figure 5-8, but three of these networks also cover the entire 360 degrees along each dimension in rotation space. This arrangement requires 3^3 sub-LLMs with a total of 216 nodes. Partitioning is only advantageous for the self-organizing map; the rigid map may also be partitioned, but this produces no improvements over a single map covering the entire rotation space. Either way, the results are disillusioning (see Figure 5-7). The prominent peaks occurring between 90 and 180 degrees are solely due to the redundancy and problematic distance measures of the roll-pitch-yaw representation as discussed in section 3.1 on page 28, which become especially evident at these ranges. The only way to alleviate these problems are more appropriate representations of pose.

5.2. 2-1-Spherical Topology

Choosing the extended spherical coordinate representation of pose is one way of overcoming the difficulties of the roll-pitch-yaw representation. It requires an external 2-1-spherical topology of the neural network, because the responses become extended spherical coordinates, and their update (2-14) must be modified accordingly (cf. section 3.2 on page 29).

The internal topology of the self-organizing map need not necessarily be changed. Since pose can always be represented by three parameters and therefore covers a three-dimensional – though not necessarily Euclidean – space, maintaining the internal cubical topology is the simplest solution. However, it was to be expected from the results reported in the previous section that such a map would have problems spreading over the entire orientation space. Again a topology-preserving map develops only if the range of views with which the network is trained is very narrow.

Therefore the next step we take is changing the internal topology of the network from cubical to 2-1-spherical as well, which is desirable because cubical neighborhood relations are certainly not ideal for a 2-1-spherical topology. In order to do this, the nodes themselves must be evenly distributed in this new space $\boldsymbol{P}$, and the distance function used to determine the neighborhood of nodes must be replaced by (3-8). Self-organizing maps of this kind show no significant improvement over the ones with internal cubical topology, though. As discussed in section 5.1.1 on page 47, the discontinuity of the input-output relation prevents a neighborhood-preserving self-organization of the nodes to a distribution that would reasonably quantize the given space.

The way out of this is the rigid map introduced in section 2.6 on page 24, which has already been demonstrated to work very effectively for cubical networks. Internal and external topology are both chosen to be 2-1-spherical, the nodes are distributed evenly on the

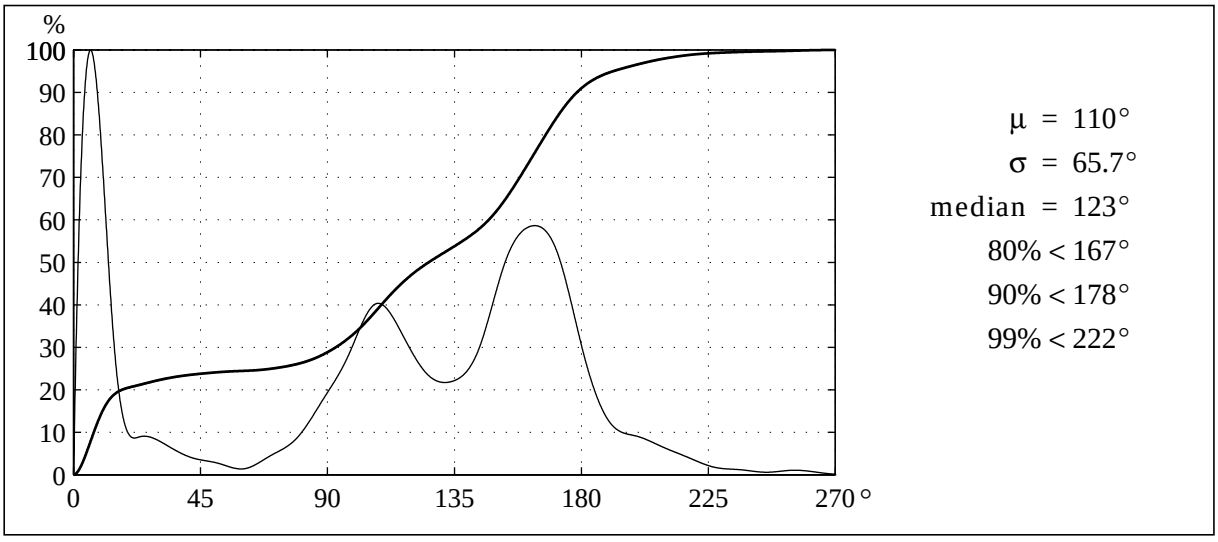


Figure 5-9: RMS error distribution of the partitioned 360° LLM. The prominent peaks between 90 and 180 degrees are due to the redundancy and problematic distance measures of the roll-pitch-yaw representation.

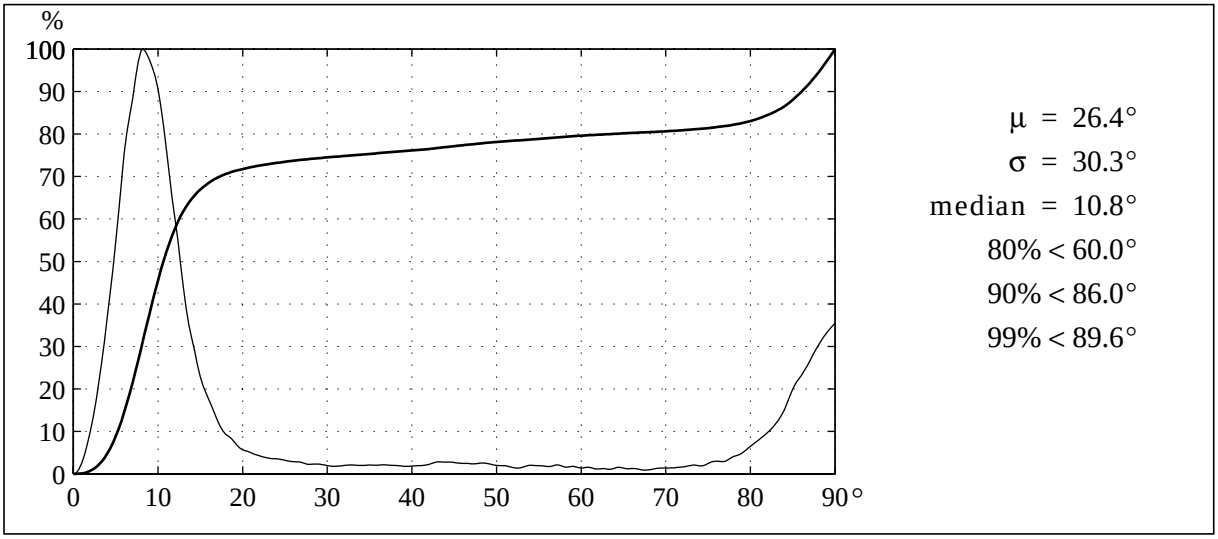


Figure 5-10: Error distribution of the rigid 2-1-spherical map with 320 nodes in the 0+ arrangement of Table 3-1. For a better comparison with the 3-hemispherical map in Figure 5-10, the quaternion error measure is used. The tail between 80 and 90 degrees is mostly due to symmetries of the tape dispenser.

2-sphere and the 1-sphere as shown in Figure 3-6 and remain at their positions throughout the (now supervised) training.

The accuracy achieved with this approach is quite good (see Figure 5-10). However, the results heavily depend on the choice of the poles of the sphere with respect to the object. It is best to position the poles in such a way that the views produced there are difficult to classify anyhow (like Figure 5-15, for example, which was chosen here as well). Otherwise “good” viewing positions, where the network would normally have no problems to give the correct pose, are ruined when the poles are positioned right there.

Unfortunately, the local linear map, which improved the accuracy of cubical networks so tremendously, fails to be effective with networks of 2-1-spherical topology, because in its original form it is limited to Euclidean vectors, as discussed in section 2.5 on page 23.

5.3. 3-Hemispherical Topology

When using the quaternion representation of pose (cf. section 3.3 on page 34), the response space $\boldsymbol{R}$ becomes 3-hemispherical. Again, the self-organized formation of a topology-preserving mapping fails. We therefore turn to the rigid map right away. Its internal and external topology are chosen to be 3-hemispherical, the nodes are distributed evenly on the 3-hemisphere and remain at their positions throughout the (now supervised) training.

For reference purposes, the best possible error distribution for a 3-hemispherical map with 360 nodes in the vc-arrangement is shown in Figure 5-11.

The accuracy achieved with this approach is very good (see Figure 5-10), in fact it is the best as yet. The improvements over the 2-1-spherical network are not overly great, but significant.

The 80% limit, i.e. the minimum accuracy sustained by 80% of the network's pose estimations, is roughly twice as low. Due to the shape of the cumulative error distribution function, this 80% limit is an extremely sensitive indicator of the quality of the network's responses and shall therefore be employed in many of the following analyses.

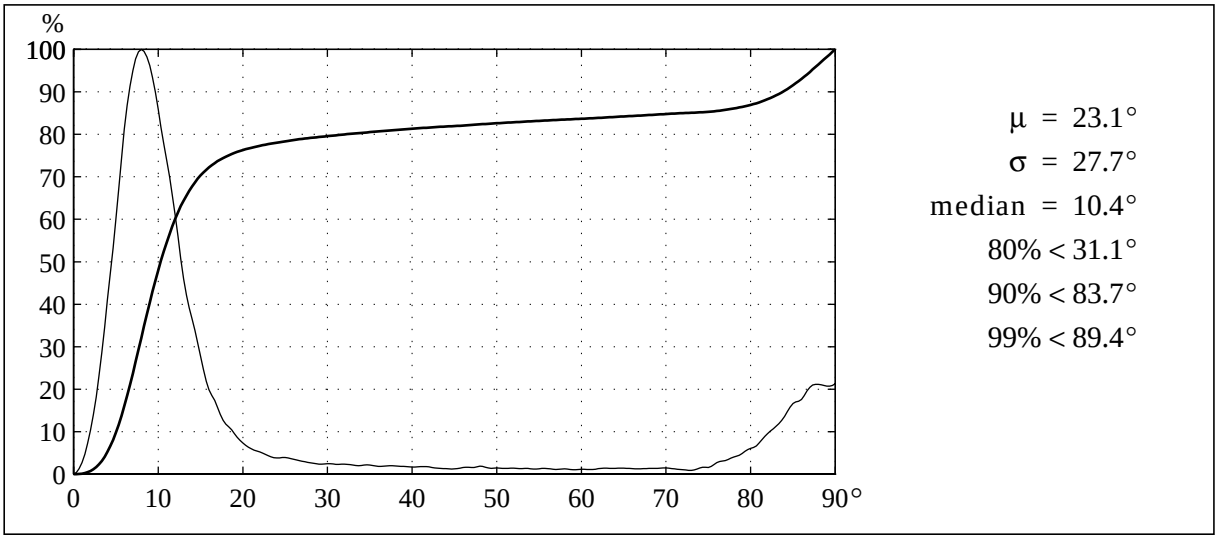


Figure 5-12: Error distribution of the rigid 3-hemispherical map with 360 nodes in the vc-arrangement of Table 3-2. The tail between 80 and 90 degrees is again mostly due to symmetries of the tape dispenser, but it is less prominent than in Figure 5-10.

5.3.1. Training Steps

It turns out that Kohonen's rule of thumb – to use at least 500 times the number of nodes in the network – is a bit too pessimistic in the case of the rigid 3-hemispherical map. In fact, as few as 50000 training steps suffice for a map with 360 nodes under these conditions, as can be

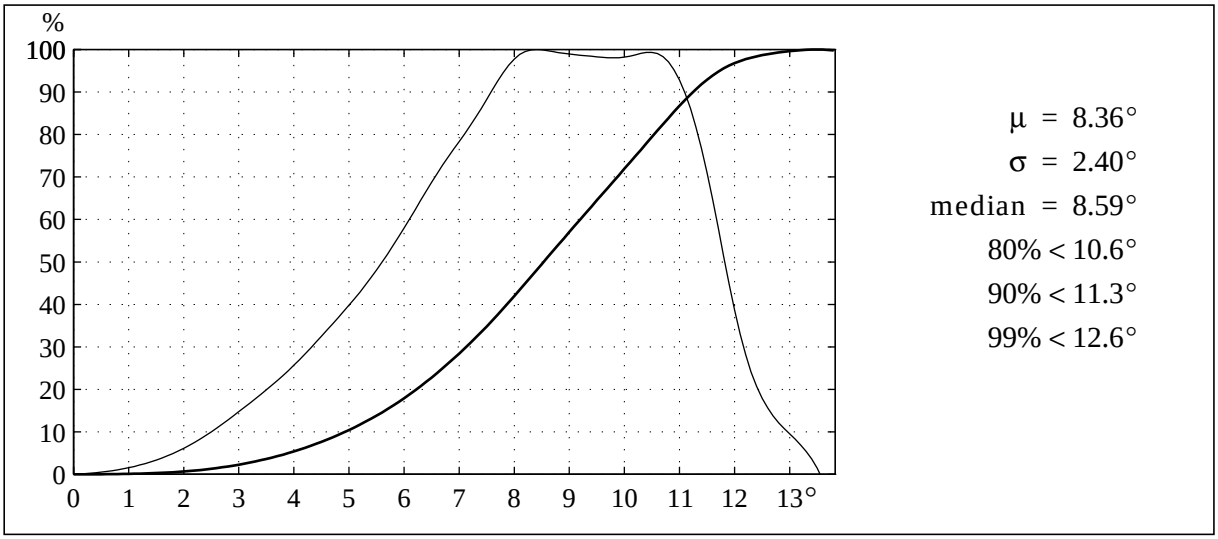


Figure 5-11: Error distribution of an ideal rigid 3-hemispherical map with 360 nodes in the vc arrangement of Table 3-2. The maximum error is approximately 13.8° .

seen from Figure 5-16. Increasing the number of training views presented to the network to a few hundred thousand produces only modest improvements.

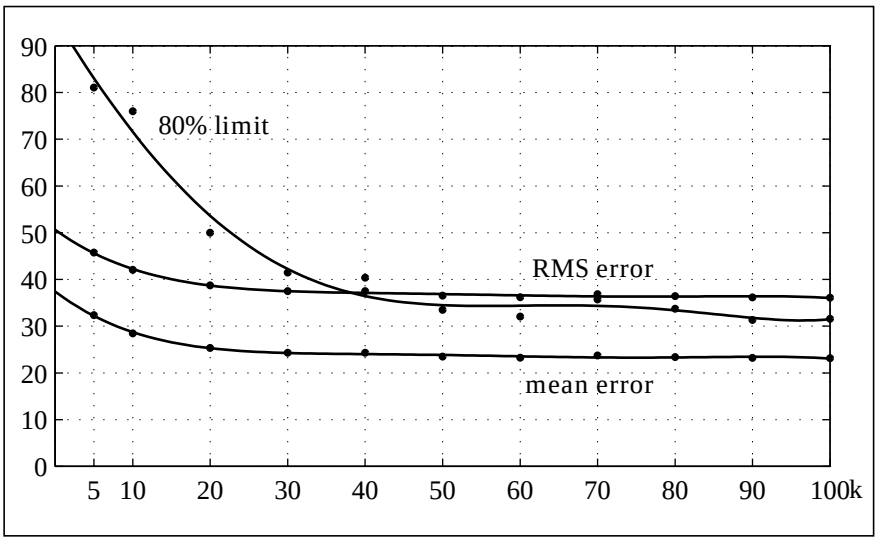


Figure 5-13: Performance over number of training steps (in thousand). Using more than 50000 training steps for a map with 360 nodes produces only modest improvements.

5.3.2. Input Dimension

From the image processing steps discussed in chapter 4, “Object Features”, result two parameters that influence the dimension of the input vector, namely the resolution of the subsampled image and the number of directions used in oriented-edge filtering. Figure 5-14 displays the effects of these parameters. The subsampled image resolution varies from 2x2 up to 16x16 pixels, and the input dimension is further increased when two or four orientations (corresponding to steps of 90° and 45° respectively) are filtered. While it was to be expected that increasing the resolution of the subsampled image could only improve results up to a certain point, it is still impressive how coarse a resolution is sufficient for the neural network. Oriented-edge filtering really only pays off for very low resolutions.

5.3.3. Hypothesis Generation

The network can also be used as a hypothesis generator whose hypotheses are tested by another algorithm, for instance inverse projection matching [62], until a good pose estimate is found. However, the process of hypothesis verification is much more expensive computation-wise than pose estimation by the neural network, so hypothesis generation should be used with care, and the number of hypotheses generated and tested should be as low as possible.

The improvements achieved with this two-step method are substantial, especially if the object has several planes of symmetry, which makes it very difficult for the neural network to decide which one is correct. In fact, an object may have aspects that represent different poses but are equivalent, in which case the correct pose cannot be determined. Take a look at Figure 5-15,

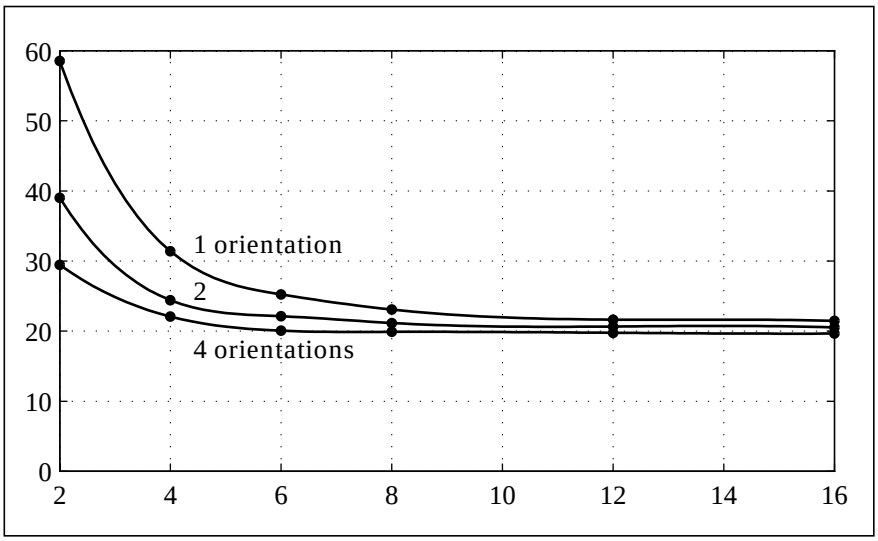


Figure 5-14: Average error over subsampled image resolution and orientations. Upon comparing input vectors of the same dimension (e.g. one orientation at a resolution of 8x8 pixels and four orientations at a resolution of 4x4 pixels), the errors are slightly lower for oriented-edge filtered images.

for instance, which shows the tape dispenser from behind – it is impossible to tell if its right or its left side is facing downwards. Actually, the tape dispenser is still a rather good-natured object in this regard.

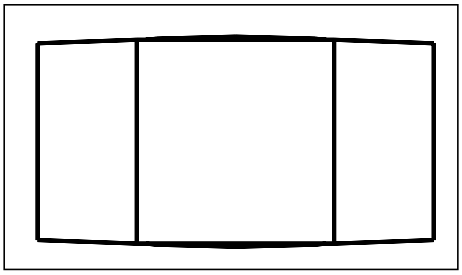


Figure 5-15: Ambiguous view of the tape dispenser. Seen from behind, it is impossible to tell if its right or its left side is facing downwards.

The generation of more than one hypothesis can be highly beneficial for the correct estimate of equally ambiguous or critical poses. Only two or three hypotheses suffice to make the pose

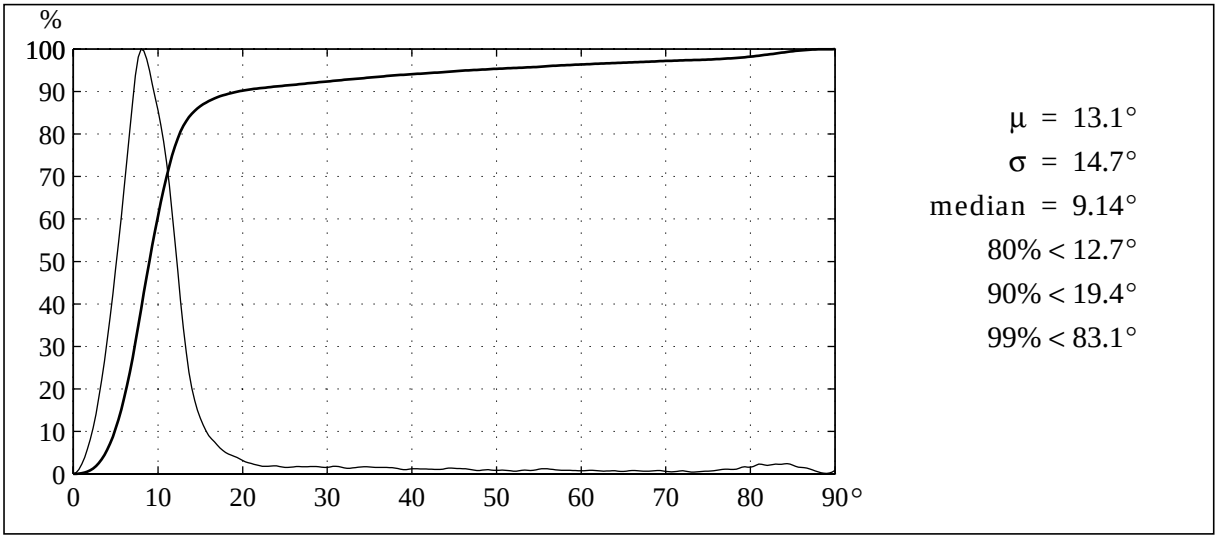


Figure 5-17: Error distribution using three hypotheses. Compared to Figure 5-10 or Figure 5-10, this distribution exhibits almost no tail, because ambiguous views of symmetric poses no longer impair the estimation results.

estimate distinctly more reliable, and five hypotheses produce near-perfect results (see Figure 5-16 and Figure 5-16).

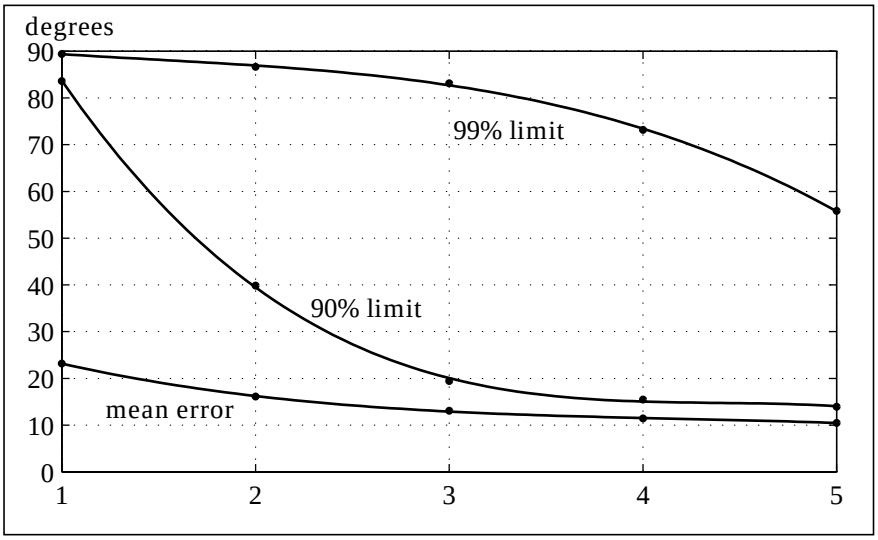


Figure 5-16: Performance over number of hypotheses. The generation of more than one hypotheses is extremely beneficial, because it eliminates estimation errors due to symmetries of the object.

5.3.4. Interpolation

Two interpolation methods were discussed in section 2.4 on page 22. Geometrical interpolation is performed between nodes close to the winner in input space. This generally

works well for self-organizing maps trained with continuous input-output relations, but fails to improve the accuracy of rigid maps. The errors are even higher than without geometrical interpolation.

Topological interpolation is performed between nodes close to the winner in response space. This approach works for both self-organizing maps and rigid maps. Even though the accuracy improvements are quite small (approximately 1 degree), they do not involve expensive computations or longer training, so there is also no reason not to use topological interpolation.

5.3.5. Varying Distance

One of the problems with multi-view representations is that the perspective projection produces different aspects of objects that have the exact same orientation, but are positioned at different distances from the camera. Lines and surfaces parallel to the line of sight become visible or invisible when the object is translated along the line of sight, unless the surfaces contain the line of sight (compare Figure 5-18).

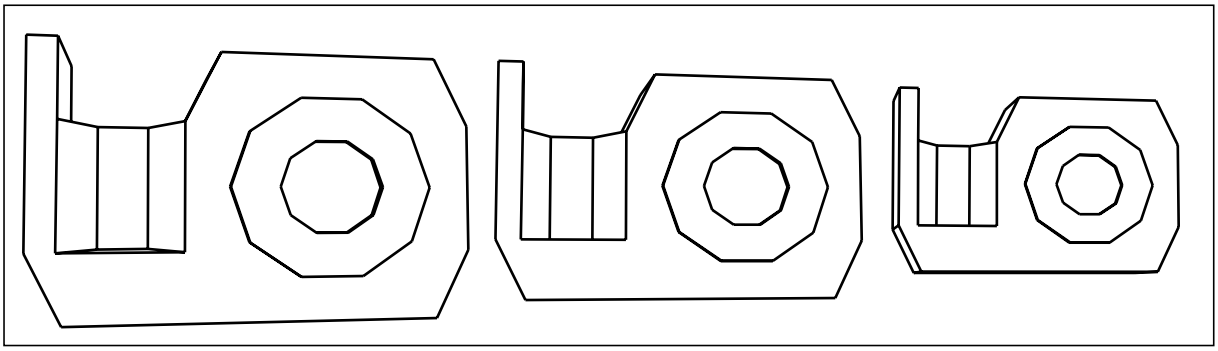


Figure 5-18: Tape dispenser at varying distances. Lines and surfaces parallel to the line of sight become visible or invisible as the object is translated along the line of sight.

It turns out that the neural network is relatively robust to these changes of aspect (see Figure 5-19). Interestingly, the errors are lowest at about 125% of the distance at which the training was performed.

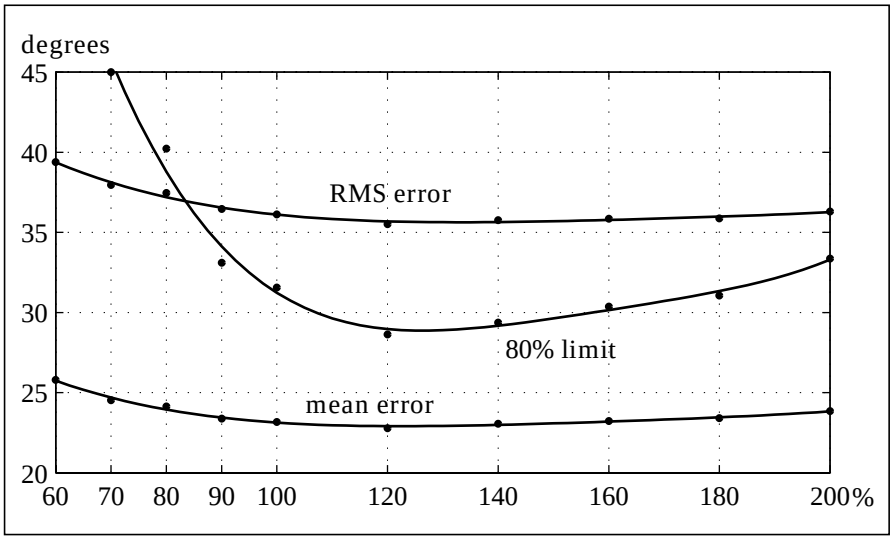


Figure 5-19: Performance with varying object distance (training distance is 100%)

5.3.6. Noise

One of the major advantages of neural networks is their graceful degradation with increasing noise in the input data. In order to demonstrate the effects of noisy input images, the vertices of the object's hidden line image are disturbed by noise of varying intensity (compare Figure 5-20). Every vertex is moved along a uniformly distributed direction by an amount of zero-mean Gaussian distribution, whose standard deviation ranges from zero to ten pixels in Figure 5-21 (the average input image diagonal measures approximately 175 pixels).

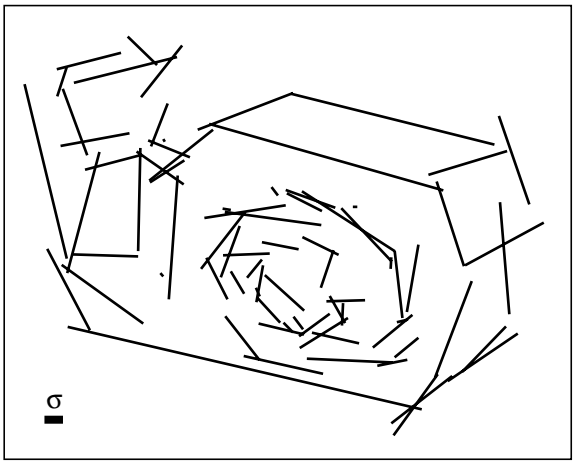


Figure 5-20: Noisy input image. Every vertex is moved along a vector, whose direction is distributed uniformly, and whose length is of zero-mean Gaussian distribution with a standard deviation of 5 pixels.

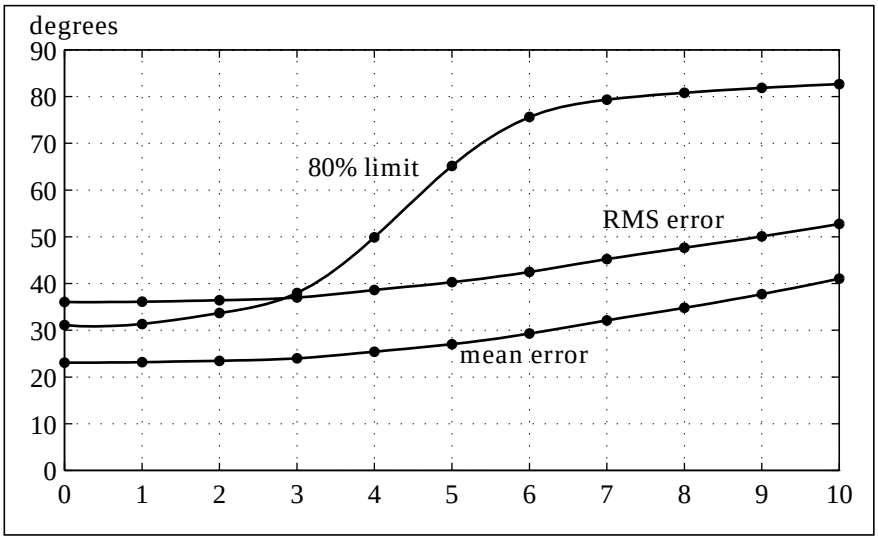


Figure 5-21: As expected, the performance of the map degrades gracefully with increasing noise (σ ranges from 0 to 10 pixels).

5.4. Real-World Input Images

Bearing in mind the application, it makes sense to take into consideration the special coordinate systems and transformations existing within the work cell of the robot illustrated in Figure 5-22. The world coordinate system **W** is the reference within the work cell. The origin of the base coordinate system **B** lies at the base of the robot. The origin of the gripper coordinate system **G** is positioned at the tool center point (TCP), and the two camera coordinate systems **C** originate in the foci of the left and the right camera respectively. The base coordinate system is fixed in relation to the world coordinate system, and so are the camera coordinate systems in relation to the TCP, because both cameras are rigidly mounted on the gripper. The transformation between **G** and **C** is determined via hand-eye-calibration [61].

Furthermore, each object in the scene has its own object coordinate system **O**. Since the object is viewed from the cameras and manipulated by the gripper, the transformations from the object coordinate system **O** to their coordinate systems **G** and **C** as well as the respective inverse transformations are of most interest.

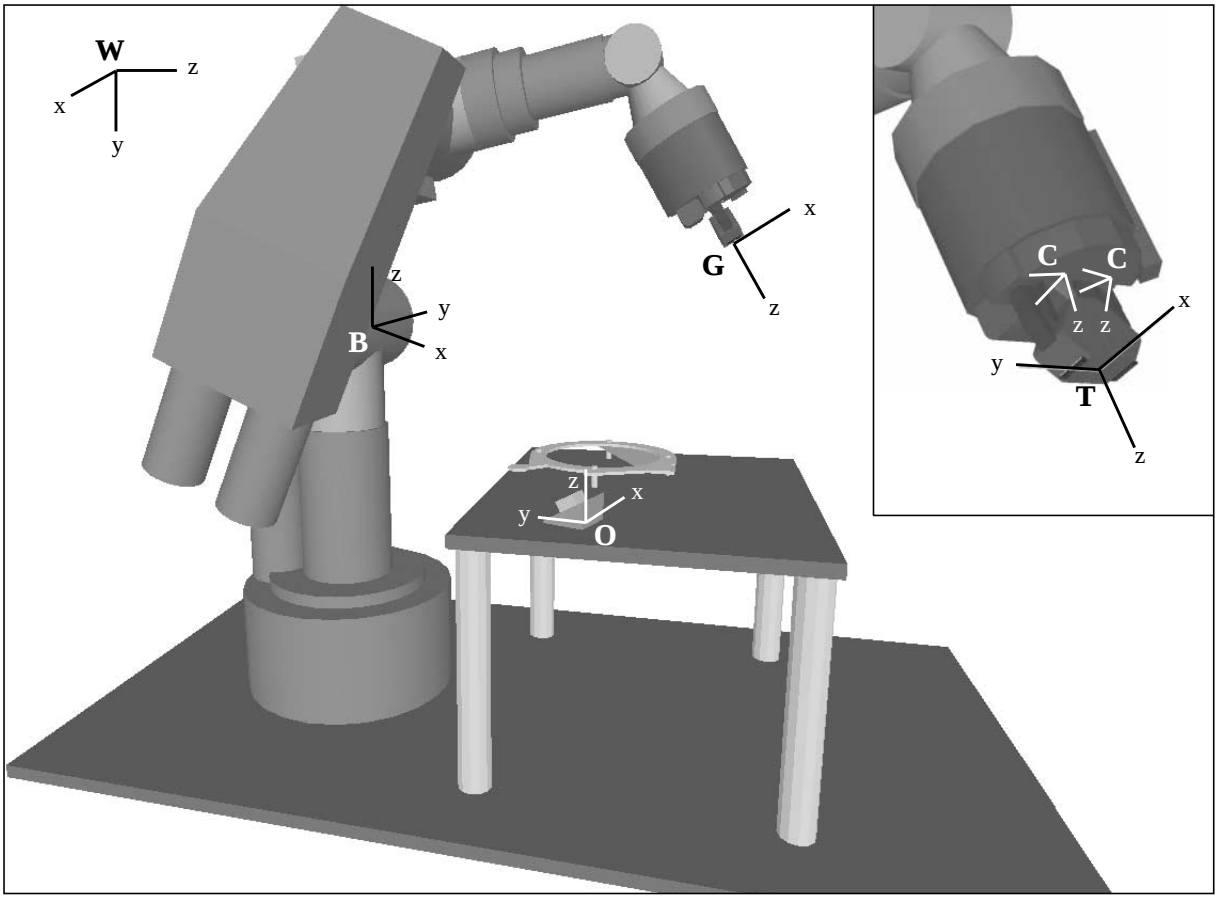


Figure 5-22: Coordinate systems in the robot's work cell

Image processing of real-world input images, including camera rectification, segmentation, oriented-edge filtering, and subsampling, is performed in real time on the Datacube.

It is much harder to obtain relevant and expressive data about the performance of the network with real-world input images than it is in the simulation. Not only would it be a formidable task to record and evaluate several thousands of uniformly distributed views, but also lighting conditions and contrast play a significant and hardly reproducible role. Therefore, the two characteristic examples in Figure 5-23 shall give an impression of the performance of the algorithm.

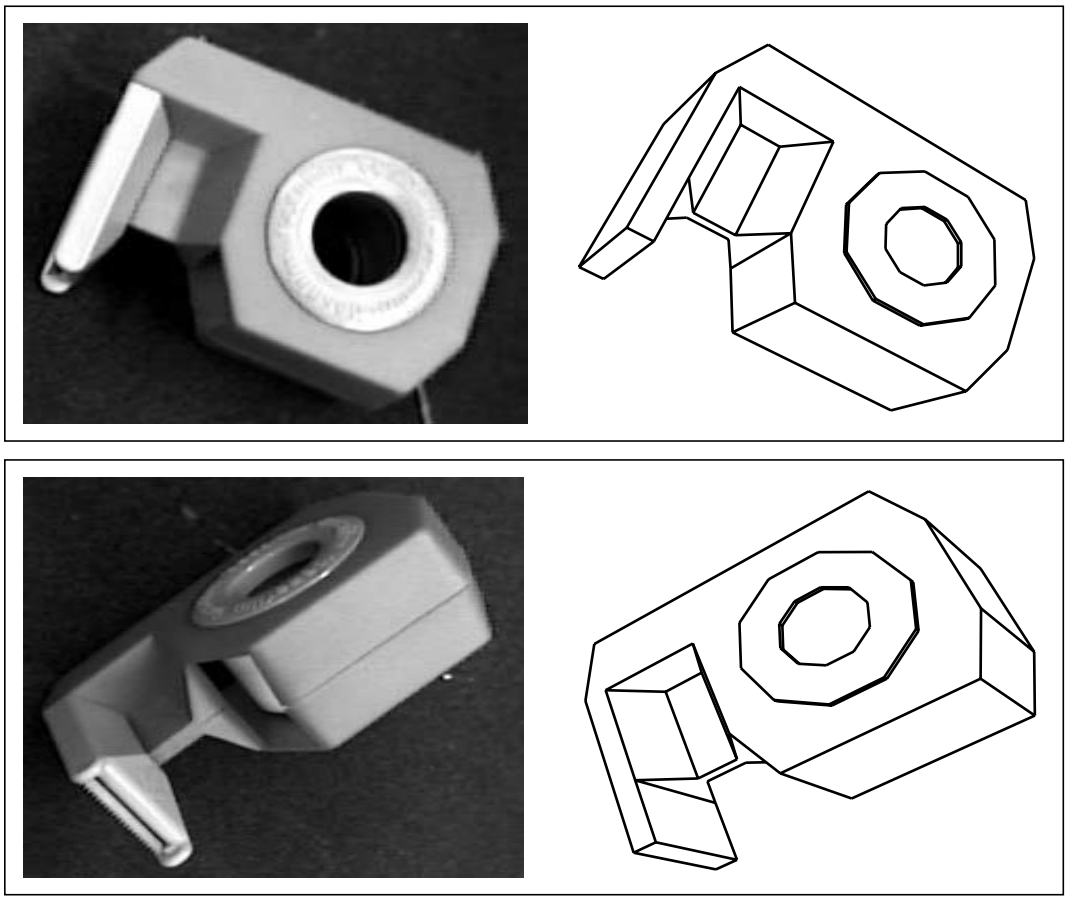


Figure 5-23: Pose estimates from real-world input images

5.5. Computational Performance

All experiments were carried out on an SGI Indigo^2 with an R4400 CPU running at 250 MHz. The image processing part (calculation of the hidden line projection, segmentation, oriented-edge filtering, subsampling etc.) consumes a lot of time in the simulation. Depending on the size and complexity of the object, producing one orientation takes approximately 10 ms. The actual training of the net (computing distances, finding the winner, updating the weights, etc.) is comparatively fast and takes approximately 20µs per node. Assuming that we increase

the number of training steps s in proportion k with the number of nodes used, as suggested by Kohonen [27], the total training time T can be summarized in the following formula:

$$T = s(t_{PP} + N t_{WU}) = kN t_{PP} + kN^2 t_{WU} . \quad (5\text{-}2)$$

Since the preprocessing time t_{PP} is about 500 times greater than the time t_{WU} it takes to update the weights of one node, the quadratic term of (5-2) will become noticeable only for networks with more than a few hundred nodes. Training time could be significantly reduced by pre-calculating a fixed set of training input vectors from maybe several thousand random views and reading them from a file instead of creating a new view for every step during training. Table 5-1 gives an overview of typical training times for some of the networks discussed in this chapter.

Table 5-1: Training times on an SGI Indigo^2 (R4400 CPU, 250 MHz)

Network specifications	Training time
LLM with 2^3 nodes, 20000 training views	3 min.
3^3 LLMs with 2^3 nodes each, 540000 training views	85 min.
320 nodes, 2-1-spherical topology, 100000 training views	35 min.
360 nodes, 3-hemispherical topology, 50000 training views	18 min.

Pose estimation alone is very fast. For real-world input images we use a Datacube, which is able to perform all of the necessary image processing steps mentioned above at least at video frame rate. Determining the winner also takes only 35µs for 360 nodes, so pose estimation in real time is possible.

"Would you tell me, please, which way I ought to go from here?"
"That depends a good deal on where you want to get to."

Lewis Carroll's Alice and the Cheshire Cat

6 Discussion

Our experiments have shown that neural networks offer an effective solution to the problem of model-based object pose estimation. While neither the original self-organizing map nor the roll-pitch-yaw representation were able to live up to our expectations, satisfactory results were achieved with rigid maps of 3-hemispherical topology implementing the quaternion representation of rotations. It may be favorable to retain self-organization, but this can only be achieved with other concepts. So-called "neural gas" approaches [10][11][40] are designed specifically to adapt to arbitrary topologies and may therefore provide a way to bring about self-organization of nodes even on the 3-hemisphere.

The correct estimation rates are very good for simulated input images, and the networks have been shown to be robust to noise as well as translations of the object with respect to the camera. Estimation errors are mostly due to symmetries of the particular object used and can be eliminated almost completely by generating more than one pose hypothesis.

Once the network has been trained, pose estimation in real time is possible. The recognition rate is determined mainly by the time required for input image processing, which can be done in real time on dedicated image processing hardware such as the Datacube.

The estimation accuracy of networks with a few hundred nodes is sufficient for use in a subsequent feature-matching algorithm. However, the accuracy could be improved even further by adapting the local linear map to non-Euclidean response spaces. As the results obtained with cubical networks suggest, the estimates could be made more precise by at least a factor of three. Alternatively, the number of nodes in the network could be reduced, leading to shorter training times.

While pose estimation from simulated input images works satisfactorily, difficulties arise upon using real-world input images. The problems are largely due to varying illumination and shadows in the camera images, which give the edge filters a hard time. Also, some edges

between regions of different colors within the object are lost, because the image processing is done with the intensity image only. All this leads to discrepancies between the modelled edges of the simulated camera image and the edge-filtered real-world input image, impairing reliable pose estimation.

Modelling the actual lighting conditions in the simulated training images would certainly improve matters. However, this is in itself a nontrivial task. Furthermore, a neural network trained under specific lighting conditions would probably not be flexible enough to cope with changing illumination, and fixed lighting conditions are difficult to maintain in practice. Other object features, possibly more elaborate than edges, may be less prone to varying lighting conditions.

One way to circumvent these problems could be training the neural network with real-world input images. While it is certainly illusory to record thousands of images suitable for training with the robot within a reasonable period of time, so many training samples may not be necessary. However, different training algorithms or altogether different network structures are required for this approach in order to sustain an acceptable level of accuracy. To give an example, parametrized self-organizing maps (PSOMs) [54][58][59] may offer a possible solution. Ideally, training need not even be done off-line but quasi on-line in just a few minutes. More investigation in these respects is definitely necessary.

I have read this book and much like it.

Moses Hadas

7 References

[1] H.-U. Bauer, T. Geisel, K. Pawelzik, F. Wolf: “Selbstorganisierende neuronale Karten” in *Spektrum der Wissenschaft*, April 1996

[2] P. Brou: “Using the Gaussian image to find the orientation of objects” in *International Journal of Robotics Research*, vol. 3, no. 4, Winter 1984

[3] G. A. Carpenter, S. Grossberg (editors): *Pattern recognition by self-organizing neural networks.* MIT Press 1991

[4] H. S. M. Coxeter: *Regular Polytopes.* 3rd edition, Dover 1973

[5] M. Dhome, M. Richetin, J.-T. Lapresté, G. Rives: “Determination of the attitude of 3-D objects from a single perspective view” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 12, December 1989

[6] D. Dunn, W. E. Higgins, J. Wakeley: “Texture segmentation using 2-D Gabor elementary functions” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 2, February 1994

[7] E. Erwin, K. Obermayer, K. Schulten: “Convergence properties of self-organizing maps” in *Proceedings of the International Conference on Artificial Neural Networks*, Elsevier Science Publishers 1991

[8] F. Fogelman Soulie: “Neural network architectures and algorithms: a perspective” in *Proceedings of the International Conference on Artificial Neural Networks*, Elsevier Science Publishers 1991

[9] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes: *Computer graphics: principles and practice.* 2nd edition, Addison-Wesley 1992

[10] B. Fritzke: “Let it grow – self-organizing feature maps with problem dependent cell structure” in *Proceedings of the International Conference on Artificial Neural Networks*, Elsevier Science Publishers 1991

[11] B. Fritzke: “Incremental learning of local linear mappings” in *Proceedings of the International Conference on Artificial Neural Networks/NeuroNîmes*, EC2 & Cie 1995

[12] R. C. Gonzalez, R. E. Woods: *Digital Image Processing.* Addison-Wesley 1992

[13] J. Göppert, W. Rosenstiel: “Topology-preserving interpolation in self-organizing maps” in *Proceedings of NeuroNîmes*, EC2 1993

[14] J. Göppert, W. Rosenstiel: “Self-organizing maps vs. backpropagation: an experimental study” in *Proceedings of the Workshop on Design Methodologies for Microelectronics and Signal Processing* 1993

[15] J. Göppert, W. Rosenstiel: “Interpolation in SOM: improved generalization by iterative methods” in *Proceedings of the International Conference on Artificial Neural Networks*, EC2 & Cie 1995

[16] W. R. Hamilton: *Elements of Quaternions.* 3rd edition, Chelsea Publishing Company 1969

[17] R. Hecht-Nielsen: “Counterpropagation networks” in *Proceedings of the IEEE International Conference on Neural Networks*, IEEE 1987

[18] G. Heidemann, H. Ritter: “A neural 3-D object recognition architecture using optimized Gabor filters” in *Proceedings of the 13th International Conference on Pattern Recognition*, IEEE 1996

[19] J. A. Hertz, A. S. Krogh, R. G. Palmer: *Introduction to the Theory of Neural Computation.* Addison-Wesley 1991

[20] B. K. P. Horn: *Robot Vision.* MIT Press 1986

[21] B. K. P. Horn: “Closed-form solution of absolute orientation using unit quaternions” in *Journal of the Optical Society of America*, vol. 4, no. 4, April 1987

[22] J. A. Kangas, T. Kohonen, J. T. Laaksonen: “Variants of self-organizing maps” in *IEEE Transactions on Neural Networks*, vol. 1, no. 1, March 1990

[23] A. Khotanzad, J. Liou: “Recognition and pose estimation of 3-D objects from a single 2-D perspective view by banks of neural networks” in *Proceedings of the Artificial Neural Networks in Engineering Conference*, ASME Press 1991

[24] H. Kim, W. Kim, C. C. Li: “A performance study of two wavelet-based edge detectors” in *Proceedings of the 11th International Conference on Pattern Recognition (C)*, IEEE 1992

[25] T. Kohonen: “Self-organized formation of topologically correct feature maps” in *Biological Cybernetics*, vol. 43, 1982

[26] T. Kohonen: *Self-Organization and Associative Memory.* 3rd edition, Springer 1989

[27] T. Kohonen: “The self-organizing map” in *Proceedings of the IEEE*, vol. 78, no. 9, September 1990

[28] T. Kohonen: “Self-organizing maps: optimization approaches” in *Proceedings of the International Conference on Artificial Neural Networks*, Elsevier Science Publishers 1991

[29] T. Kohonen: “What generalizations of the self-organizing map make sense?” in *Proceedings of the International Conference on Artificial Neural Networks*, Springer 1994

[30] P. Koikkalainen, E. Oja: “Self-organizing hierarchical feature maps” in *Proceedings of the International Joint Conference on Neural Networks*, IEEE 1990

[31] P. Koikkalainen: “Fast deterministic self-organizing maps” in *Proceedings of the International Conference on Artificial Neural Networks/NeuroNîmes*, EC2 & Cie 1995

[32] J. Langwald: *Bildgestützte Lagebestimmung dreidimensionaler Objekte mittels Aspektgraphen.* DLR-Institutsbericht 515-95-16, Diplomarbeit, TU Ilmenau 1995

[33] R. P. Lippmann: "An introduction to computing with neural nets" in *IEEE Acoustics, Speech, and Signal Processing Magazine,* April 1987

[34] R. P. Lippmann: "Pattern classification using neural networks" in *IEEE Communications Magazine*, November 1989

[35] M.-C. Lu, C.-H. Lo, H.-S. Don: "A neural network approach to 3-D object identification and pose estimation" in *Proceedings of the International Joint Conference on Neural Networks*, IEEE 1991

[36] S. P. Luttrell: "Self-organizing multilayer topographic mappings" in *Proceedings of the IEEE International Conference on Neural Networks*, IEEE 1988

[37] C. Maggioni, B. Wirtz: "A neural net approach to 3-D pose estimation" in *Proceedings of the International Conference on Artificial Neural Networks*, Elsevier Science Publishers 1991

[38] S. G. Mallat, W. L. Hwang: "Singularity detection and processing with wavelets" in *IEEE Transactions on Information Theory*, vol. 38, no. 2, March 1992

[39] T. M. Martinetz, H. J. Ritter, K. J. Schulten: "Three-dimensional neural net for learning visuomotor coordination of a robot arm" in *IEEE Transactions on Neural Networks*, vol. 1, no. 1, March 1990

[40] T. M. Martinetz, K. J. Schulten: "A 'neural-gas' network learns topologies" in *Proceedings of the International Conference on Artificial Neural Networks*, Elsevier Science Publishers 1991

[41] R. Mehrotra, K. R. Namuduri, N. Ranganathan: "Gabor filter based edge detection" in *Pattern Recognition*, vol. 25, no. 12, December 1992

[42] A. Meyering, H. J. Ritter: "Learning 3-D shape perception with local linear maps" in *Proceedings of the International Joint Conference on Neural Networks*, IEEE 1992

[43] K. R. Miller, J. F. Gilmore: "Object recognition using neural networks and high-order perspective-invariant relational descriptions" in *SPIE Intelligent Robots and Computer Vision X*, vol. 1607, SPIE 1991

[44] G. Nagy: "Candide's practical principles of experimental pattern recognition" in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 5, no. 2, March 1983

[45] Y. H. Pao: *Adaptive Pattern Recognition and Neural Networks.* Addison-Wesley 1989

[46] K. Park, D. J. Cannon: "Recognition and localization of a 3D polyhedral object using a neural network" in *Proceedings of the International Conference on Robotics and Automation*, IEEE 1996

[47] T. Peter: *Effizientes Eliminieren verdeckter Kanten in polyedrischen Szenen zur Simulation von Kamerabildern.* DLR-Institutsbericht 515-96-5, Diplomarbeit, FH Regensburg 1996

[48] K. Pfreunder: *Kamerageregeltes Positionieren eines Roboters.* DLR-Institutsbericht 515-96-1, Diplomarbeit, TU München 1996

[49] T. Poggio, S. Edelman: "A network that learns to recognize three-dimensional objects" in *Nature*, vol. 343, January 1990

[50] H. J. Ritter, K. J. Schulten: "Kohonen's self-organizing maps: exploring their computational capabilities" in *Proceedings of the IEEE International Conference on Neural Networks*, IEEE 1988

[51] H. J. Ritter: “Combining self-organizing maps” in *Proceedings of the International Joint Conference on Neural Networks*, Lawrence Erlbaum Associates 1989

[52] H. J. Ritter, T. M. Martinetz, K. J. Schulten: *Neuronale Netze.* Addison-Wesley 1990

[53] H. J. Ritter: “Learning with the self-organizing map” in *Proceedings of the International Conference on Artificial Neural Networks*, Elsevier Science Publishers 1991

[54] H. J. Ritter: “Parametrized self-organizing maps for vision learning tasks” in *Proceedings of the International Conference on Artificial Neural Networks*, Springer 1994

[55] W. S. Sarle (editor): *FAQ of the comp.ai.neural-nets Usenet newsgroup*, 1996

[56] K. Shoemake: “Animating rotation with quaternion curves” in *SIGGRAPH ‘85 Conference Proceedings, ACM Computer Graphics*, vol. 19, no. 3, July 1985

[57] V. Tresp: “Die besonderen Eigenschaften neuronaler Netze bei der Approximation von Funktionen” in *Künstliche Intelligenz*, May 1995

[58] J. Walter, H. J. Ritter: “Local PSOMs and Chebyshev PSOMs improving the parametrized self-organizing maps” in *Proceedings of the International Conference on Artificial Neural Networks/NeuroNîmes*, EC2 & Cie 1995

[59] J. Walter, H. J. Ritter: ““Rapid learning with parametrized self-organizing maps” in *Neurocomputing, Special Issue* 1996

[60] B. Widrow, M. A. Lehr: “30 years of adaptive neural networks: perceptron, madaline and backpropagation” in *Proceedings of the IEEE*, vol. 78, no. 9, September 1990

[61] P. Wunsch, G. Koegel, K. Arbter, G. Hirzinger: *Calibration of a non-linear eye-in-hand system.* Technical Report, DLR 1995

[62] P. Wunsch, G. Hirzinger: “Registration of CAD-models to images by iterative inverse perspective matching” in *Proceedings of the 13th International Conference on Pattern Recognition* 1996

[63] P. Wunsch: *Multisensoriell gestütztes Greifen unter Verwendung von CAD-Modellen.* Dissertation, TU München (in preparation)

When I use a word, it means just what I choose it to mean,
neither more nor less.

Lewis Carroll's Humpty Dumpty

8 Notation

Unless mentioned otherwise, the symbols and acronyms listed below have been used with the following meaning:

i	node index
LLM	local linear map
$\mathbf{L}_i$	local transformation matrix of LLM-node
M	number of input weights, dimension of the input vector
N	number of nodes in the network
$\boldsymbol{p}_i$	position of node i (internal topology)
$\boldsymbol{P}$	position space (internal topology)
r_x, r_y, r_z	rotation about x-, y-, z-axis (yaw, pitch, roll)
$\hat{\boldsymbol{r}}$	input response
$\tilde{\boldsymbol{r}}$	overall network response
$\boldsymbol{r}_i$	response of node i (external topology)
$\boldsymbol{R}$	response space (external topology)
RLLM	rigid local linear map
RM	rigid map
RMS	root mean square
SOLLM	self-organizing local linear map
SOM	self-organizing map
t_x, t_y, t_z	translation along x-, y-, z-axis
w	index of winner

$\hat{\mathbf{w}}$	input weight vector
$\mathbf{w}_i$	weight vector of node i
δ, δ_S, δ_C	general, spherical and circular distance measures
ε_{RMS}	root mean square error
λ	learning rate
μ	statistical average
ν	neighborhood function
σ	standard deviation
ω	angular wrap-around function