Doctoral Thesis ETH No. 16896

Spike-based Winner-Take-All Computation in a Multi-Chip Vision System

A dissertation submitted to the

Swiss Federal Institute of Technology Zurich

for the degree of

Doctor of Technical Sciences

presented by

Matthias Ferdinand Oster

Diplom-Ingenieur Univ., Technische Universität München

born 13. Juni 1977
citizen of Germany

accepted on the recomendation of

Prof. Dr. Rodney J. Douglas, examiner
Dr. Shih-Chii Liu, co-examiner
Prof. Dr. Bernabe Linares-Barranco, co-examiner

2006

Summary

The asynchronous and time-continuous computation that takes place in biological systems is of great interest because of the capability of these systems to interact with the real-world. In this work we explore such computation in the spike-based winner-take-all network, by developing a theoretical model of its performance and describing its implementation in a large-scale multi-chip vision system.

The winner-take-all is a neuronal network that amplifies the strongest set of inputs and suppresses output from the others. In various neuroscience models, this function is used to make a selection out of a possible set of decisions dependent on the input. As we develop artificial spike-based systems for different applications we need a theoretical understanding of the parameter settings in the spike-based neuronal networks where we can obtain this behavior. Previous analyses of winner-take-all behavior have considered analog or spike-rate coded inputs. Here we are interested in the computation in the transition between single-spike and spike rate coding.

In the theoretical part of this thesis (Chapter 2), we show for both regular and Poisson spike trains under which parameters the decision of the winner is optimal, that is it makes use of all information available in the input spikes. For inputs of regular rates the winner can be selected with only one inter-spike interval. For inputs of Poisson rates, the performance of the network depends on the number of spikes the neurons need to reach threshold.

In biology, the winner-take-all network is believed to be part of the cortical microcircuit, in which it both cooperates and competes with other areas to reach a consistent interpretation of the input. We model such context information with self-excitation if the input is stationary, with permitted sets to embed syntactic constraints, and with competition across winner-take-all networks to select the strongest of a set of feature maps.

We then describe how these theoretical analyses are applied in Very-Large-Scale-Integration technology (Chapter 3). Neurons and synapses are implemented using analog transistor circuits, while spikes are represented as digital address-events. We focus on two issues in the implementation: mismatch and a hybrid hardware/software framework. First, mismatch in the analog transistors limits the performance of the winner-take-all network. We characterize the

mismatch in the input synapses and discuss different schemes of compensating the synaptic weights. Second, we present a software framework to analyze the spike trains in the system and to add learning and adaptation capabilities by embedding software agents in the spiking communication.

Finally, in Chapter 4, we analyze the hardware winner-take-all in a multi-chip vision system (CAVIAR), consisting of an artificial retina, a bank of spike-based convolution filters, the winner-take-all network and a learning module that classifies trajectories of moving objects. We show that the inputs to the winner-take-all follow Poisson statistics, although retina and convolution filters are deterministic and exhibit only a small amount of variation. When the system is stimulated with a moving object, the input to the winner-take-all is a spike wave of Gaussian shape that is traveling across the neurons. With this input we can compare the winner-take-all performance predicted by our theoretical model with the performance of our implementation when applied to this large-scale system that performs real-time asynchronous computation.

Zusammenfassung

Die asynchrone und zeitkontinuierliche Informationsverarbeitung biologischer Systeme ist von großem Interesse, da sie diesen Systemen ermöglicht, in Echtzeit mit der Welt zu interagieren. In dieser Arbeit erforschen wir diese Rechenprinzipien in einem pulsbasierten neuronalen Netz zur Maximumselektion. Wir entwickeln ein theoretisches Modell der Leistungsfähigkeit und beschreiben die Implementierung in einem hochintegrierten Bildverarbeitungssystem.

Neuronale Netze zur Maximumselektion verstärken die stärkste Gruppe von Eingangssignalen und unterdrücken die Aktivität der anderen Neurone. Diese Funktion wird häufig in neurowissenschaftlichen Modellen verwendet, um Entscheidungen in Abhängigkeit des Eingangs auszuwählen. Um solche pulsbasierten Netzwerke in technische Applikationen umzusetzen, benötigen wir ein theoretisches Verständnis ihrer Wirkungsweise und Parameterbereiche. Vorhergehende Untersuchungen von Netzwerken mit Maximumselektion haben die Eingänge entweder als analoge Signale oder aber frequenzkodiert angesehen. Unser Interesse gilt den Prinzipien der Berechnung im Übergang der Kodierung als einzelne Pulse und als Pulsrate.

Im Theorieteil dieser Dissertation (Kapitel 2) untersuchen wir die Parameterbereiche des Netzwerks für die optimale Erkennung von Eingangssignalen mit konstanter Rate und mit Poisson-Statistik. Wir sprechen von optimaler Erkennung, wenn alle in den Eingangssignalen enthaltene Informationen ausgenutzt werden. Bei konstanter Rate kann das Netzwerk das stärkste Eingangssignal mit nur zwei Pulsen detektieren; bei Kodierung mit Poisson-Statistik hängt die Erkennungsleistung von der Anzahl der Pulse ab, die die Nervenzellen zum Erreichen ihres Schwellwertes benötigen.

Selektionsnetzwerke werden in der Modellierung von kortikalen Standardnetzwerken eingesetzt, um Informationen verschiedener Gehirnareale gleichzeitig kompetitiv und kooperativ zu integrieren. Wir berücksichtigen solche Kontextinformationen in unserem Netzwerk durch Selbsterregung der Neurone für stationäre Eingangssignale, durch Koaktivierungsmuster für die Integration syntaktischer Nebenbedingungen, und durch Selektion auf Netzwerkebene in der Musterkennung mit Filterbänken.

Das theoretische Verständnis des Netzwerks ermöglicht die Umsetzung als hochintegrierte Halbleiterschaltung (Kapitel 3). Neurone und Synapsen werden

durch analoge Transistorschaltkreise implementiert; digitale Pulse modellieren die Aktionspotentiale biologischer Nervenzellen. Wir konzentrieren uns auf zwei Implementierungsfragen: auf Parameterschwankungen und die Integration von Algorithmen. Parameterschwankungen in den analogen Schaltkreisen mindern die Leistungsfähigkeit des Selektionsnetzwerks. Wir untersuchen die Parameterschwankungen in den Eingangssynapsen und diskutieren verschiedene Verfahren zur Kompensation der synaptischen Gewichte. Als zweites entwickeln wir eine hybride Hardware/Softwarearchitektur zur Analyse der Pulsfolgen im System, und um Lern- und Adaptionsalgorithmen in die Pulskommunikation zu integrieren.

In Kapitel 4 diskutieren wir die Leistungsfähigkeit des Netzwerkes eingebettet in ein hochintegrierte Bildverarbeitungssystem (CAVIAR), bestehend aus einer künstlichen Retina, einer pulsbasierten Filterbank, dem Selektionsnetzwerk und einem Lernmodul, das die Trajektorien sich bewegender Objekte klassifiziert. Wir zeigen, daß die Signale im System der Poissonverteilung folgen, obwohl sowohl Sensor als auch die Vorverarbeitung deterministisch sind und nur geringe Parameterschwankungen aufweisen. Sich bewegende Objekte lösen eine Welle von Aktivität im Profil einer Normalverteilung aus, die die Neuronen des Netzes entlangläuft. Mit dieser Modellierung der Eingangssignale kann die Leistungsfähigkeit des Netzwerkes in der Vorhersage des theoretischen Modells mit der Anwendung in einem hochintegrierten System verglichen werden, um echtzeitfähige asynchrone Rechenarchitekturen zu verstehen.

Contents

Chapter 1

Introduction

"Biological information processing systems ('brains') overwhelmingly outperform their artificial counterparts."[1]

This citation from the CAVIAR project description underscores the fact that our current artificial computation systems, as powerful as they are when processing large data sets, still fail miserably when compared to the interaction capability of biological systems with the real world. Taking inspiration from key principles of biological information processing systems to improve our artificial systems is a logical step.

Many differences between biological and artificial computational architectures have been pointed out, such as the complex recurrent connectivity of neurons in cortex, the co-localization of processing and local memory, or the adaptive properties of biological wet-ware. In this work we will emphasize another key principle that forms the basis for interaction in real-time: the incorporation of time into the basic building block of computation.

Modern processing and signal processing architectures execute software on synchronously clocked hardware. At each clock cycle, the input signals are considered to be static, and are combined in boolean functions. The duration of the clock cycle is fixed, determined by the maximum propagation delay of the underlying functions. The underlying building block, the gate, is seen as a quasi-static element, as well as the functional description of its logic function. Similarly, most descriptions of time-varying signals in engineering depend on constant sampling frequencies, for example the sampling of sensory signals or the fourier frequency analysis.

Neurons as the basic building blocks of biological computation do not compute based on a clock but perform their computation continuously in time and asynchronously on the input. Computation is triggered by incoming spikes (events), and performed with functions defined on different time-scales. The

[1] CAVIAR, EU 5^{th} framework project (IST-2001-34124), project description p.4.

result of the computation is again sampled asynchronously by transmitting output spikes to other neurons. Basically, the processing in the neuron is triggered by the input signals and reflects the input timing. We think that this incorporation of time into the basic building block of computation is the basis for the interaction of the whole system with the real-time world.

The same basic building block, the neuron, is repeated on all levels of abstraction. Throughout the hierarchy of processing, spikes can encode sensory information, extracted features, behavior decisions and motor reactions.

The type of mathematics needed to describe such asynchronous, event-based processing is different from conventional engineering methods, with sometimes surprising results. For example, sampling at Poisson-distributed timepoints leads to an alias-free frequency representation [Shapiro and Silverman, 1960].

We work towards an understanding of asynchronous event-based processing by quantifying the performance of a simple spike-based neuronal network, a winner-take-all. This network computes a MAX operation with respect to the statistical properties of its inputs. We will reduce the network to a maximally simplified, minimalistic form that captures the essence of this spike-based computation, before we add biologically relevant parameters and quantify their effect on the network behavior (Chapter 2). Having understood the basic principles, we describe an implementation of the network in Very-Large-Scale-Integration (VLSI) technology (Chapter 3). Embedding this building block in a large-scale multi-chip system in the CAVIAR project (Chapter 4) leads the way towards processing architectures that are inspired by computation principles found in biological systems such as cortex.

1.1 Winner-Take-All Networks

The human brain consists of roughly 10^{10} neurons, of which each receives connections from up to 10^4 other neurons. The number of connectivity patterns that are possible with these numbers is beyond imagination and would render any understanding of cortical processing impossible. But the cortex shows a structure of six different layers, and the different layers show different cell types and different connectivity patterns, preserved over the cortex. This structure has given rise to the proposition that there is a fundamental neuronal circuit, the so-called canonical microcircuit [Douglas et al., 1989]. This circuit specifies a connectivity pattern of different cell types in the cortical layers and its input from the thalamus. Being repeated many times in each cortical area, the canonical microcircuit would be one of basic network-level building blocks of the brain, and its deciphering is the key to an understanding of cortical structure.

Unfortunately, the properties of this microcircuit make it difficult for researches to obtain an integral snapshot of its network: its size is below the resolution of magnetic resonance imaging and positron emission tomography, and it has too many neurons for a complete anatomical reconstruction or recording

from multiple cells. Two-photon microscopy would offer the resolution to image such a network, but its imaging planes are parallel to cortex surface, while the main connections of the neurons in the network extend vertically. Significant effort has been undertaken over the last years to create a statistical map of the neuronal circuit by detailed anatomical reconstruction of connections between single cells [Binzegger et al., 2004]. In this statistical map, the relations of connections between the cell types in the different layers are listed. At the same time, the functional significance of this microcircuit structure is explored by simulations in which the ratios of neuron types and connections match the ones found by anatomy [Douglas et al., 1989, Korner et al., 1999, Maass et al., 2002, 2004].

[Douglas and Martin, 2004] derive a model of cortical function from the anatomical data as follows: superficial pyramidal cells receive input from subcortical areas, as well as from other intra- and interareal excitatory sources. Horizontal inhibitory cells ensure that the local input is decoded consistently with the interpretation of other intra- and inter-areal networks. This selection mechanism is implemented by a soft winner-take-all network that cooperates to enhance consistent interpretations of the input, and competes to suppress inconsistent ones. Built on this stage of feature decoding is a second structure that has a similar soft selection configuration to further process the decoded information and to decide which output is transmitted to the motor structures.

While the actual processing might be very specific to the function the microcircuit is used in, the winner-take-all network is a very general neuronal selection circuit. Several studies have discussed the computational power it offers [Riesenhuber and Poggio, 1999, Yuille and Geiger, 1998, Maass, 1999, 2000]. It has been applied to many models of cortical processing, for example to a hierarchical model of vision in cortex [Riesenhuber and Poggio, 1999], or to model selective attention and recognition processes [Itti et al., 1998].

What are the properties of the winner-take-all selection mechanism? The winner-take-all amplifies the strongest set of inputs and suppresses output from the other neurons. The winner-take-all is 'hard' if only one neuron is selected as the winner and all other neurons are suppressed. In contrast, we use the term 'soft selection' if several neurons are active in the output. There are several properties that are motivated by the response of neurons in the cortex. These properties combine digital and analog signal processing [Douglas et al., 1995, Hahnloser et al., 2000]:

Soft selection: in contrast to selecting only one neuron as the winner, multiple neurons are active in the output. In cortex-like implementations this fuzzy selection is desirable for biological plausibility. We will show in Section 2.2 that co-activation of other neurons than the winner can emerge from the variation found in the coding of cortical input signals.

Linear amplification: the activity of the winner is proportional to its input. In contrast, a digital process would select the winner without the output

signaling any information about the strength of its input [Douglas et al., 1995]. In our model, the output frequency of the winner is an integer division of the input frequency, so linearity is an intrinsic property.

Gain modulation: the modulation of the output response by background input activity. The output is amplified depending on the total amount of input the network is receiving. This can be seen as a kind of neuronal multiplication [Salinas and Abbott, 1996] and is obtained in network models based on linear threshold units and firing rate input [Hahnloser et al., 2000, Dayan and Abbott, 2001]. For our spike-based implementation without lateral connectivity, gain modulation is not an intrinsic network property and is not necessary in our desired application. Gain modulation could be added by introducing additional lateral connectivity across the input.

Sustained activity: the persistence of the selection, that is, the output persists even after the input is taken away or becomes uniform [Dayan and Abbott, 2001, pp. 256]. This phenomenon is related to hysteresis, the resistance of the network to follow a change in the strongest input [Hahnloser et al., 2000]. Our model exhibits hysteresis on a single spike based level (Section 2.3.1), but not on the level of a sustained spiking output without any input signal.

Signal restoration: the network restores the stimulus to a pattern that is latent in the recurrent connectivity. Patterns can be simple nearest-neighbor interaction or more complex patterns in which groups of neurons are active. We will discuss this model of permitted and forbidden sets as defined by [Hahnloser et al., 2000] in Section 2.5.1. These networks are comparable with the concept of attractors, see [Dayan and Abbott, 2001, pp265].

Because of these properties the winner-take-all circuit is a smart decision element as part of the microcircuit. Decision processes in the brain are not localized in one specific region, but evolve in a distributed manner when different brain regions cooperate to reach a consistent interpretation. The winner-take-all circuit with both cooperative and competitive character is a main building block which contributes to this distributed decision process.

Because of this functionality, the winner-take-all computation has been of large interest to researchers. [Yuille and Grzywacz, 1989] and [Ermentrout, 1992] are classical references to theoretical analyses. In these early models, the neurons are non-spiking, that is they receive an analog input and have an analog output.

This analog winner-take-all computation can be efficiently implemented in VLSI transistor circuits. With the initial circuits described in [Lazzaro et al., 1989], a whole series of analog models, for example [Kaski and Kohonen, 1994, Barnden and Srinivas, 1993], and implementations has been developed over the last years [He and Sanchez-Sinencio, 1993, Starzyk and Fang, 1993, Serrano

et al., 1994, Kincaid et al., 1996, Indiveri, 1997, Moller, 1998, Liu, 2000b, Indiveri, 2001, Liu, 2002].

In the last decade, spiking neuron models have gained increasing interest. Neuron models with analog inputs and analog outputs can be converted into models with spiking output if a thresholding operating is introduced to the neuron. [Coultrip et al., 1992] is an early theoretical analysis, with further descriptions in [Jin and Seung, 2002, Yu et al., 2002] and VLSI implementations in [Indiveri et al., 2002, Chicca et al., 2004, Abrahamsen et al., 2004].

The next consideration are models with both spiking input and spiking output. Theoretical analysis is focused on population models (for example [Lumer, 2000]), in which the population firing represents a graded analog value. [Indiveri et al., 2001, Chicca et al., 2006] are VLSI implementations that use the firing rate as an analog input and output encoding. A special case is [Carota et al., 2004, Bartolozzi and Indiveri, 2004], in which the core of the winner-take-all is analog, but the signals are converted to spike rates for communication with the outside world. Other theoretical analyses consider other neuron models, such as oscillatory neurons [Wang, 1999] or differentiating units [Jain and Wysotzki, 2004].

To our knowledge, no analysis until now has considered the effect of single spikes and spike timing on the winner-take-all computation with spiking inputs and outputs. This is the scope of the analysis presented in this thesis in Chapter 2. We will explore the computation in the transition between single-spike and rate coding.

[Gautrais and Thorpe, 1998] start their analysis from a similar point of view, that is the distinction which of two input spike rates is higher, but they do not consider sampling of this estimation in the output spikes (we could classify this analysis as spiking input and analog output).

Having understood the basic principles of the winner-take-all operation in theory (Chapter 2), we can approach an implementation of the winner-take-all network on chip in Section 3.2. Comparable to the variations found in biological neurons and synapses, analog VLSI circuits exhibit variation because of mismatch in the transistor characteristics. Mismatch changes the performance of analog VLSI winner-take-alls. We will characterize the mismatch on the chip in Section 3.3 and discuss compensation procedures. Second, we will discuss how the adaptive properties of biological systems can be incorporated in the implementation using software algorithms (Section 3.4).

1.2 Hardware Vision Architectures

Early neuromorphic implementations have focused on simple applications and architectures. An example is the processing of visual motion, with chips that extract motion information from the environment. We compared these three implementations [Liu, 2000a, Higgins et al., 2005, Stocker, 2006] in the 'Senso-

ryIntegration' project[2] [Ortelli, 2004]. Motion processing in analog VLSI implementations seems especially suitable for two reasons. First, the computation in these models, which are partly inspired by the fly visual system, is only analog and does not require high-level information. Second, motion processing in conventional computer architectures requires a lot of resources since two or more subsequent imager frames have to be stored and compared. However, since the chips have to integrate additional circuits into the pixel, the fill factor that can be obtained by these imagers is lower compared to chips that only integrate the photoreceptor. Furthermore, it can be difficult to optimize the intermediate stages of the computation (for example, the delay lines in case of a Reichhard detector) since not all the intermediary results can be accessed externally. A different approach, therefore, is to separate the functional modules into separate building blocks. [Ozalevli and Higgins, 2005] and [Indiveri et al., 1999] are examples of motion processing systems in which the imager and the post-processing have been separated into individual chip implementations. This, of course, comes with an increased overhead in terms of infrastructure, since the intermediate results have to be encoded for communication between the chips.

All sensors working on real data are subject to noise. The effect of this noise can be decreased by integrating more global information. [Stocker, 2006] uses a resistive grid to smooth the local motion information. [Yang et al., 2006] propagates predicted information together with a moving edge across the chip, following an algorithm by [Woergoetter et al., 1999]. Being able to reliably detect information about the real-world seems to require more complex computation than just processing local features. The design and test of more complex algorithms therefore requires more flexible system architectures than a single chip.

The CAVIAR project develops a vision processing framework in which all functional modules are separated into individual building blocks. Each building block is defined by one particular function, for example the computation of the temporal derivative in case of the retina. Every building block can therefore be designed and optimized as a standalone module. All modules use the same AER protocol definition, so different architectures can in principle be assembled to test algorithms of larger complexity. The performance is increased by combining the development effort of four labs, in which everyone contributes a building block. The CAVIAR project can be seen as an attempt to establish a common infrastructure for multi-chip systems.

This modular approach is one of the main differences of the interfaces of the CAVIAR project to its predecessor, the Silicon Cortex (SCX) project [Deiss et al., 1999]. SCX incorporated all functionality into one main module which in result was too complex for the development effort of one working group. Nevertheless, SCX laid out the foundation for a multi-module computing architecture from which the CAVIAR project could use prior knowledge.

[2] 'SensoryIntegration' project, funded by the Institute of Neuromorphic Engineering (INE), Maryland, U.S.

In the next paragraphs, we will compare the CAVIAR system to other architectures that implement biologically-inspired computation and vision processing. This is the HMAX network for biological plausibility, SpikeNET for its spike-coding scheme, and in general feature extraction and classification systems.

The HMAX network is inspired by biologically plausible models of connectivities in the network for vision processing. Along the processing hierarchy, spikes encode more and more information about the detected object. Spatial invariance is achieved by pooling over the field of view. In Section 4.1 we will give a detailed description and comparison of the HMAX network and the CAVIAR implementation.

Stating that the communication is spike-based as in CAVIAR does not make any assumptions about the specific coding scheme used in the transmission. While traditionally the spike rate is used to encode the signal strength, other schemes have also been proposed. [Van Rullen and Thorpe, 2001, VanRullen and Thorpe, 2002] propose a visual information processing system, SpikeNET, in which signals are encoded in a rank order code. Such a code provides an efficient coding, since every neuron transmits only one spike. The information rate per spike is therefore higher than in rate coding approaches [Gautrais and Thorpe, 1998, Thorpe and Gautrais, 1998]. Such an encoding uses different neuronal networks for decoding and learning [Perrinet et al., 2001, Delorme et al., 2001]. However, some of the building blocks are similar to CAVIAR, for example a convolution filter and a network with fast inhibition [Delorme, 2003]. The single spike coding scheme was shown to be efficient in the application of visual processing in software [Thorpe et al., 2004, Delorme et al., 1999]. The flexible architecture of CAVIAR could be used to implement such an efficient coding scheme and to explore its functional performance in a hardware system that interacts with the real-world.

The convolution stage of CAVIAR is used to extract features from the input. Many feature extractors have been developed for technical systems. Most artificially designed features are well optimized for a specific task, but fail to show good performance when applied to general tasks (for example the principal axis projection method, see [Yagi and Shibata, 2002, Yagi et al., 2002]). Programmable features like the convolution kernels offer the freedom to adjust the feature filters to the application. Such an architecture is the system from [Bandyopadhyay et al., 2005], which provides programmable matrix transformations to the output of an imager.

In biology, a hierarchy of features of increasing complexity is used. At the beginning of this hierarchy in primary visual cortex, neurons are mostly selective to orientation and spatial frequency. A hardware model that describes the extraction of orientation is [Choi et al., 2005], based on the output of the imager in [Choi et al., 2004]. CAVIAR can implement orientation detection by programming simple orientation filters as convolution kernels. However, how a hierarchy of features with increasing complexity is constructed remains an open

problem, a discussion which we briefly touch in Section 4.1.

Once the features are extracted they have to be classified. While the CAVIAR system uses only a limited number of features and classifies these in a two-stage hierarchy of winner-take-alls, non-spiking systems often extract large numbers of features from the input. These features are then combined in a multi-dimensional vector. From the huge field of implementations of classification algorithms available in the literature, we want to name three examples relevant for neuromorphic implementations. Radial basis functions [Madani et al., 2003] provide a classification that is similar to the tuning curves found in the receptive fields of neurons. Vector-quantizers implemented on-chip are [Ogawa et al., 2002, Hasler et al., 2002]. [Serrano-Gotarredona and Linares-Barranco, 1996, 1997] describe implementations of the ART algorithm, a self-organizing neural pattern recognition machine developed in the 1990s.

More higher-level algorithms like the principal component analysis or the Fisher discriminant are cluster algorithms that can be used to extract features with high information content for classification. These algorithms could be used to identify interesting features about the input which can then be programmed as convolution kernels into the CAVIAR system. However, these methods regard the input data as static and do not consider the dynamic properties of the inputs as CAVIAR does.

In this work we explore these dynamic properties of the input spike trains to CAVIAR. In Section 4 we will describe the CAVIAR system in more detail. The CAVIAR system provides an ideal opportunity to explore spike-based processing with the winner-take-all network in a large-scale vision processing system. The amount of spiking data that can be obtained through the system is unique: no recording technique is able to record data from such an amount of single neurons in biology. We will therefore develop data processing methods to assess the spike train statistics, and finally to quantify the winner-take-all performance in Section 4.4.

Chapter 2

Theory of Winner-Take-All Operation

In this chapter we analyze the functionality of a spike-based winner-take-all network. We see the network as an event-based classifier. The network has to decide which neuron, the "winner", receives the strongest input after a certain time interval, and it indicates its decision with an output spike. In a spiking system, the term 'strongest input' has to be defined: how is the input signal encoded in the spikes? What are the statistical properties of the spike trains? We consider the following cases: stationary inputs of regular frequencies, stationary inputs of Poisson distribution, the switching of the strongest input between two neurons, and a model of non-stationary Poisson inputs in which the strongest input is traveling across the neurons of the network.

In the case of spiking inputs of regular frequencies, we show that the performance of the network can be optimal, that is the network completely suppresses all activity except that of the winning neuron. For this case we discuss the boundary conditions under which the hard winner-take-all behavior takes place.

In the case of inputs of Poisson distribution, the performance of the winner-take-all scales with the number of spikes the neurons needs to reach threshold. For short decision times, that is a small number of input spikes, the variation in the Poisson input leads to variation in the output. For longer decision times, the network averages over more input spikes, leading to a more precise decision. We derive a formula to quantify this performance. With this formalization, we can examine the effect of network parameters like the strength of mutual inhibition, or self-excitation. We validate our formula with simulation data; this also allows us to quantify the effect of leakage in the soma of the neurons.

We then extend our analysis to non-stationary inputs with time-varying input spike rates. We answer the question whether a cascade of winner-take-all networks with short integration times is more efficient than a winner-take-all

network with long integration times. We conclude that the network outputs should be as sparse as possible, that is a single spike as output of the network is optimal for classification. We use this criterion to analyze a case of non-stationary Poisson inputs in which the input is a Gaussian wave traveling across the neurons. As we will show in Chapter 4, this is a good model of the CAVIAR processing chain that computes the center of a moving object in the field of view. We define an error function to quantify how well the stimulus location is predicted by the winner-take-all network. In this chapter we compare the functional description with simulation results, before we will look at the results from the implementation in the CAVIAR system in Chapter 4.

In the last section of this chapter, we discuss how context information can be embedded in the winner-take-all network through the local excitatory connections, by embedding structure in the recurrent connectivity, and through competition on the next level, across winner-take-all networks.

2.0.1 Connectivity

We assume a network of integrate-and-fire neurons that receives external excitatory or inhibitory spiking input. To implement a hard winner-take-all operation, these neurons compete against one another through inhibition. In biological networks, excitation and inhibition are specific to the neuron type. Excitatory neurons make only excitatory connections to other neurons and inhibitory neurons make only inhibitory connections. Inhibition between the array neurons is always mediated by populations of inhibitory interneurons (Figure 2.1, a). The inhibitory neurons are driven by the excitatory neurons, and in return they inhibit the excitatory neurons.

To adjust the amount of inhibition between the neurons (and thereby the strength of the competition), both types of connections could be modified: the excitatory connections from array neurons to interneurons and the inhibitory connections from interneurons to array neurons. In our model, we assume the forward connections between the excitatory and the inhibitory neurons to be strong, so that each spike of an excitatory neuron triggers a spike in the global inhibitory neurons. The amount of inhibition between the array neurons is adjusted by tuning the connections from the global inhibitory neurons to the array neurons. This configuration allows the fastest spreading of inhibition through the network and is consistent with findings in biology that the inhibitory interneurons tend to fire at high frequencies.

With this configuration, we can simplify the network by replacing the global inhibitory interneurons with full inhibitory connectivity between the array neurons (Figure 2.1, b). This simplification is only used during the analysis; for implementation the configuration with the global interneurons is more suitable.

Neurons in cortex integrate input from up to 10000 synaptic connections per neuron. Since we do not consider correlation effects between single inputs, we can summarize all input in one input signal by adding up all firing rates. This

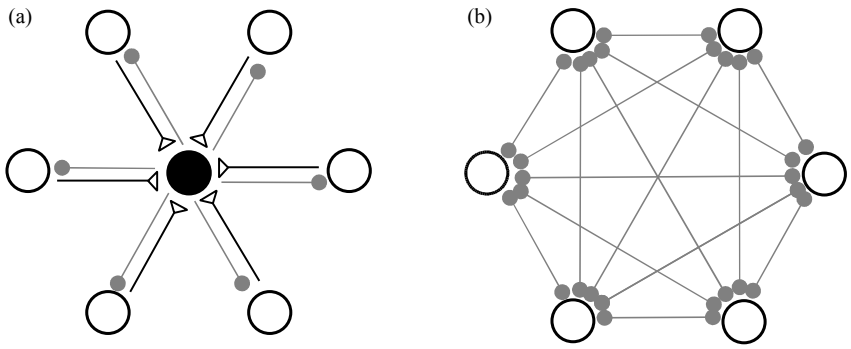

Figure 2.1: Simplification of connectivity: (a) with a global inhibitory neuron; (b) with direct inhibitory connections between neurons.

assumption is especially valid for Poisson-distributed input, since combining several spike trains with Poisson statistics results again in Poisson statistics.

In addition to the external input and the inhibitory synapses, each neuron has a self-excitatory synapse that facilitates the selection of this neuron as the winner in the next cycle once it has been chosen as the winner. Additional connectivity that implements context information will be subject to discussion in Section 2.5.

We do not consider the inhibitory inputs to the array neurons. Inhibitory input could be added by assigning a sign to every input spike. Since the statistical properties stay the same, we disregard the external inhibitory input for clarity.

2.0.2 Neuron Model

We assume a network of N integrate-and-fire neurons $i = 1 \ldots \mathrm{N}$. The membrane potentials V_i satisfy the equation of a non-leaky integrate-and-fire neuron model with non-conductance-based synapses:

$$\frac{\mathrm{d}V_i}{\mathrm{d}t} = V_E \sum_n \delta(t - t_i^{(n)}) - V_I \sum_{\substack{j=1 \\ j \neq i}}^{N} \sum_m \delta(t - t_j^{(m)}) \tag{2.1}$$

$t_i^{(n)}$ denotes the time of the nth spike to neuron i. The $\delta(t)$ is the delta function defined as 1 at $t=0$ and 0 otherwise.

The membrane resting potential of the neurons is 0. Each neuron receives external excitatory input and inhibitory connections from all other neurons. All inputs to a neuron are spikes, and its output is also transmitted as spikes to other neurons. We neglect the dynamics of the synaptic currents and the delay

in the transition of the spikes. Each input spike causes a fixed discontinuous jump in the membrane potential (V_E for the excitatory synapse and V_I for the inhibitory). Each neuron i spikes when $V_i \geq V_{th}$ and is reset to $V_i = 0$. Immediately afterwards, it receives a self-excitation of weight V_{self}. All potentials satisfy $0 \leq V_i \leq V_{th}$, that is, an inhibitory spike can not drive the membrane potential below ground.

2.1 Stationary Inputs of Regular Rates

We first discuss the case where the neurons receive spike trains of regular frequency as inputs. Considering a non-leaky integrate-and-fire neuron model, the network will select the winning neuron after receiving a pre-determined number of input spikes. The winning neuron is the one receiving the input with the smallest inter-spike interval.

All neurons $i \in 1 \ldots \mathrm{N}, i \neq k$ receive excitatory input spike trains of constant frequency r_i. Neuron k receives the highest input frequency ($r_k > r_i \; \forall \, i \neq k$).

As soon as neuron k spikes once, it has won the competition. Depending on the initial conditions, other neurons can at most have transient spikes before the first spike of neuron k. For this hard winner-take-all mode, the network has to fulfill the following constraints (Figure 2.2):

(a) Neuron k (the winning neuron) spikes after receiving $n_k = n$ input spikes that cause its membrane potential to exceed threshold. After every spike, the neuron is reset to V_{self}:

$$V_{self} + n_k V_E \geq V_{th} \tag{2.2}$$

(b) As soon as neuron k spikes once, no other neuron $i \neq k$ can spike because it receives an inhibitory spike from neuron k. Another neuron can receive up to n spikes even if its input spike frequency is lower than that of neuron k because the neuron is reset to V_{self} after a spike, as illustrated in Figure 2.2. The resulting membrane voltage has to be smaller than before:

$$n_i \cdot V_E \leq n_k \cdot V_E \leq V_I \tag{2.3}$$

(c) If a neuron j other than neuron k spikes in the beginning, there will be some time in the future when neuron k spikes and becomes the winning neuron. From then on, the conditions (a) and (b) hold, so a neuron $j \neq k$ can at most have a few transient spikes.

Let us assume that neurons j and k spike with almost the same frequency (but $r_k > r_j$). For the inter-spike intervals $\Delta_i = 1/r_i$ this means $\Delta_j > \Delta_k$. Since the spike trains are not synchronized, an input spike to neuron k has a changing phase offset ϕ from an input spike of neuron j. At every output spike of neuron j, this phase decreases by $\Delta\phi = n_k(\Delta_j - \Delta_k)$ until $\phi < n_k(\Delta_j - \Delta_k)$. When this

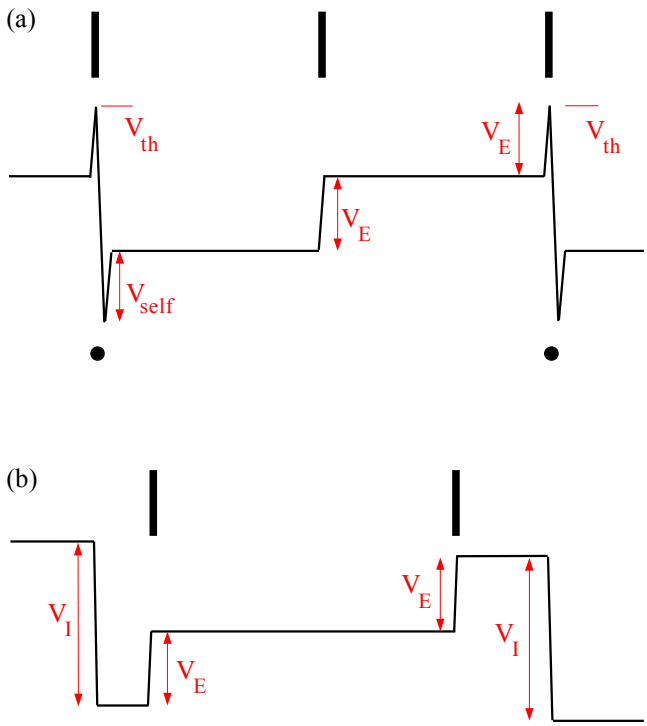

Figure 2.2: Membrane potential of the winning neuron k (top) and another neuron in the array (bottom). Black bars show the times of input spikes. Traces show the changes in the membrane potential caused by the various synaptic inputs. Black dots show the times of output spikes of neuron k.

happens, neuron k receives (n_k+1) input spikes before neuron j spikes again and crosses threshold:

$$(n_k + 1) \cdot V_E \geq V_{th} \tag{2.4}$$

We can choose $V_{self} = V_E$ and $V_I = V_{th}$ to fulfill the inequalities (2.2)-(2.4). V_E is adjusted to achieve the desired n_k.

Case (c) happens only under certain initial conditions, for example when $V_k \ll V_j$ or when neuron j initially received a spike train of higher frequency than neuron k. A leaky integrate-and-fire model will ensure that all membrane potentials are discharged ($V_i = 0$) at the onset of a stimulus. The network will then select the winning neuron after receiving a pre-determined number of input spikes and this winner will have the first output spike.

If the rate is regular, the information about the strongest input is already contained in one inter-spike-interval. If $V_{th}/2 < V_E < V_{th}$ and $V_{th}/2 < V_{self} < V_{th}$ is chosen, the network also performs the selection in one inter-spike-interval. We call this an optimal decision, since the network exploits all information in the input. The performance of the network is as good as the information available.

The case (c) can be interpreted as hysteresis: if the strongest input switches from neuron k to neuron j, the network exhibits hysteresis depending on the difference in the frequencies.

Due to the spike-timing, the mechanism of competition is independent from the number of neurons. The network can be scaled to any size, as long as the inhibitory neuron can still completely inhibit the array neurons with one output spike. Other models that exploit firing rate thresholds are normally dependent to the number of neurons in the network. The performance of the network decreases also in the case of inputs of Poisson statistics, as we will show in the next section.

In the case of perfect network homogeneity, the network can detect the winner optimally. In Section 3.3, we will discuss how variation in the synaptic parameters leads to a decrease in performance.

2.2 Stationary Poisson Rates

In the case of Poisson-distributed spiking inputs, there is a probability associated with the selection of the correct winner. This probability depends on the Poisson rate ν and the number n of spikes needed for the neuron to reach threshold. We first analyse the winner-take-all network with a simple example of only two neurons labeled '0' and '1', with the connectivity shown in Figure 2.3.

The Poisson distribution is given by the probability that n spikes arrive in time t for rate ν:

$$\mathrm{P}(\nu t, n) = \frac{(\nu t)^n}{n!} \exp(-\nu t) \tag{2.5}$$

We assume that in the initial state the neurons are completely discharged ($V = 0$). The neurons cross threshold at time t if they receive n–1 input spikes in $[0; t[$

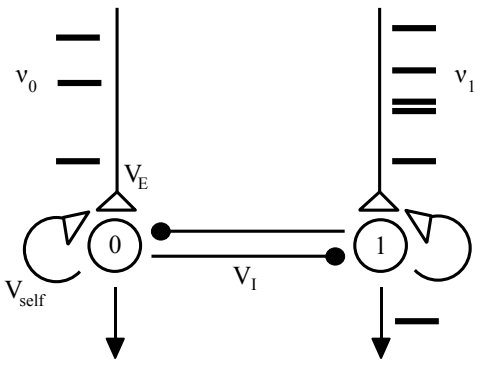

Figure 2.3: Simplified network architecture. Two neurons labeled '0' and '1' receive input spike trains ν_0 and ν_1 of Poisson distribution. The synaptic strengths are V_E for the excitatory input, V_I for the cross-inhibition and V_{self} for the self-excitation.

and the nth one exactly at time t. The first term is given by the probability $P(n-1, \nu t)$, the second probability is the rate ν itself.

Neuron '0' reaches threshold at time t with the probability density

$$p_0(t) = \nu_0 \mathrm{P}(\nu_0 t, n-1) \tag{2.6}$$

Neuron '1' does not reach threshold until time t if it receives fewer or equal than $n-1$ spikes in $[0; t[$:

$$P_{\bar{1}}(t) = \sum_{i=0}^{n-1} \mathrm{P}(\nu_1 t, i) \tag{2.7}$$

Neuron '0' has the first output spike of the network if it reaches threshold at time t (probability density p_0), while neuron '1' does not reach threshold (probability $P_{\bar{1}}$). We integrated over all times $t = 0 \ldots \infty$:

$$P_{0out} = \int_0^\infty \mathrm{P}(\nu_0 t, n-1) \cdot \nu_0 \cdot \left(\sum_{i=0}^{n-1} \mathrm{P}(\nu_1 t, i) \right) \mathrm{d}t \tag{2.8}$$

We first discuss the properties of Equation 2.8 in the extreme cases. For $n = 1$, every input spike elicits an output spike. The probability that the first output spike of the network comes from neuron '0' is the higher rate normalized to the sum of both rates:

$$P_{0out}\Big|_{n=1} = \int_0^\infty \nu_0 e^{-\nu_0 t} e^{-\nu_1 t} \mathrm{d}t = \nu_0 \int_0^\infty e^{-(\nu_0+\nu_1)t} \mathrm{d}t = \frac{\nu_0}{\nu_0 + \nu_1} \tag{2.9}$$

For $n \to \infty$ or very large n, the neurons integrate over many input spikes. The Poisson distributed rates can then be approximated by the mean rate and the decision is deterministic:

$$P_{0out}\Big|_{n\to\infty} \to \begin{cases} 1 & : \quad \nu_0 > \nu_1 \\ 0 & : \quad \nu_0 < \nu_1 \end{cases} \tag{2.10}$$

For $n \to \infty$ the winner-take-all network would of course never produce an output spike since it has to integrate over an infinite number of input spikes.

In the general case, the probability that the first spike of the network comes from the neuron receiving the higher input rate increases as the neurons integrate over more spikes. Equation 2.8 can not be solved symbolically, but can be integrated numerically.

Until now we discussed the probability of the first spike the network elicits. As soon as one neuron spikes, the other neuron is inhibited. We assume strong inhibition ($V_I = V_{th}$) and no self-excitation ($V_{self} = 0$), so that both neurons are completely discharged after the first output spike. The neuron that spiked is reset by its reset mechanism, the other neuron by the inhibition it receives. The network then has no memory of its past input and integration will start again with the initial conditions. The probability P_{0out} that the first spike of the network comes from neuron '0' is then equivalent to the probability that *any* output spike of the network comes from neuron '0', if $V_I = V_{th}$ and $V_{self} = 0$.

P_{0out} is independent of the absolute rate. To show this, we replace $\nu_0 \to P_{0in}\nu$ and $\nu_1 \to (1 - P_{0in})\nu$ with $\nu = \nu_0 + \nu_1$. We substitute $\nu t \to t'$ and $\nu \, \mathrm{d}t \to \mathrm{d}t'$. The integration limits $(0; \infty)$ do not change.

$$P_{0out} = \int_0^\infty \mathrm{P}(P_{0in}t', n-1) \cdot P_{0in} \cdot \left(\sum_{i=0}^{n-1} \mathrm{P}((1 - P_{0in})t', i) \right) \mathrm{d}t' \tag{2.11}$$

In this formulation the winner-take-all is a filter that increases the percentage of spikes encoding for neuron '0'. Figure 2.4 details this interpretation as a filter model in signal theory.

The total input rate is a Poisson spike train of rate ν. The output rate ζ is the sum of the output rates of each neuron. To make an output spike, the neurons integrate over n input spikes. Neuron '0' receives input spikes with rate νP_{0in}, neuron '1' with rate $\nu(1-P_{0in})$. The sum is weighted by the probability that neurons '0' and '1' make an output spike (P_{0out} and $1-P_{0out}$).

$$\begin{aligned} \zeta &= P_{0out}\frac{\nu P_{0in}}{n} + (1 - P_{0out})\frac{\nu(1 - P_{0in})}{n} && (2.12) \\ &= \frac{\nu}{n}\left(1 - P_{0in} - P_{0out} + 2P_{0in}P_{0out}\right) && (2.13) \end{aligned}$$

The output spike train is not Poisson distributed, but will tend towards regular rates, since the neurons integrated over n input spike for each output spike.

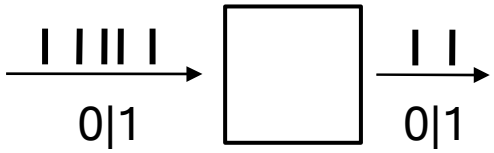

Figure 2.4: Filter model of the winner-take-all network. The winner-take-all (black box) receives a Poisson input spike train of total rate ν. Every spike encodes either for symbol '0' or '1', with the probability of a spike coding for '0' of P_{0in}. The output of the winner-take-all is a spike train with the probability of a spike coding for '0' of P_{0out}. The winner-take-all ensures $P_{0out} > P_{0in}$ and amplifies the difference between input and output probabilities.

We quantify the effect of the winner-take-all on the input and output probabilities in Figure 2.5. The optimal classification would result in a step function: for any $P_{0in} > 0.5$ the output probability P_{0out} is 1. The output performance increases if the neurons integrate over more input spikes n. We compare the results of our functional description with simulation results, to verify both functional description and simulation routines.

In the general case ($V_I < V_{th}$ and $V_{self} \neq 0$), the assumption that both neurons are completely discharged after an output spike does not hold anymore. The neuron that spiked is reset to ground and receives self-excitation V_{self}. It will reach threshold again with m spikes:

$$m = \left\lceil \frac{V_{th} - V_{self}}{V_E} \right\rceil \leq n \tag{2.14}$$

The neuron that did not spike is inhibited by the spiking neuron, which lowers its membrane potential by V_I. It will reach threshold again with p spikes, dependent on the membrane potential at the time (t^-) of inhibition.

$$p = \left\lceil \frac{V_{th} - \max(V(t^-) - V_I, 0)}{V_E} \right\rceil \leq n \tag{2.15}$$

$V(t^-)$ can take any value between ground and just below the threshold voltage. We will assume that the non-winning neuron needs p spikes to reach threshold again. In Section 2.2.2 we will analyze if this assumption is valid.

We model the output of the network by a Markov model with inhomogeneous transition times with two states (Figure 2.6). The states determine which of the two neurons fired last. Each transition from state i to j has assigned a transition probability p_{ij} and a transition time t_{ij}. In contrast to a Markov chain, the states are only sampled when the network spikes and the transition times between the states are not constant.

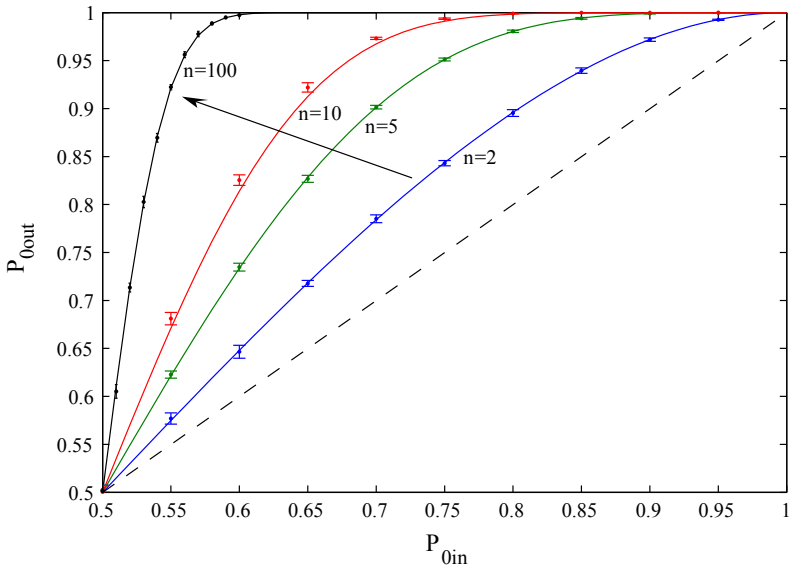

Figure 2.5: Probability of neuron '0' having an output spike of the network, with strong inhibition and no self-excitation. P_{0in} is the probability of a spike encoding for neuron '0' in the input rate; P_{0out} is the probability of a spike encoding for neuron '0' in the output rate. The different curves show the dependence on the number of input spikes n the neurons are integrating over.The probability that the first output spike of the network comes from the neuron that receives the higher rate increases if the neurons integrate over more input spikes. The output probabilities are independent of the input rate ν. Continuous curves: numerical evaluation of Equation 2.11, overlaid data points: simulation results (error bar corresponds to 10 trials with 10000 output spikes each).

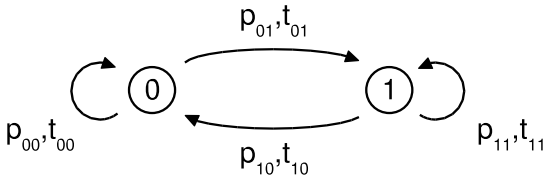

Figure 2.6: Markov model of the winner-take-all network with inhomogeneous transition times: the states determine which of the two neurons fires. The transition probabilities p_{ij} together with the transition time t_{ij} determine the sequence of firing.

The transition probabilities are given by:

$$p_{00} = \int_0^\infty \nu_0 \, \mathrm{P}(\nu_0 t, m-1) \cdot \sum_{i=0}^{p-1} \mathrm{P}(\nu_1 t, i) \, \mathrm{d}t \tag{2.16}$$

$$p_{01} = \int_0^\infty \sum_{i=0}^{m-1} \mathrm{P}(\nu_0 t, i) \cdot \nu_1 \, \mathrm{P}(\nu_1 t, p-1) \, \mathrm{d}t \tag{2.17}$$

$$p_{10} = \int_0^\infty \nu_0 \mathrm{P}(\nu_0 t, p-1) \cdot \sum_{i=0}^{m-1} \mathrm{P}(\nu_1 t, i) \, \mathrm{d}t \tag{2.18}$$

$$p_{11} = \int_0^\infty \sum_{i=0}^{p-1} \mathrm{P}(\nu_0 t, i) \cdot \nu_1 \, \mathrm{P}(\nu_1 t, m-1) \, \mathrm{d}t \tag{2.19}$$

The transition time is the number of spikes a neuron needs to reach threshold divided by its input rate:

$$t_{00} = \frac{m}{\nu_0}, \quad t_{01} = \frac{p}{\nu_1}, \quad t_{10} = \frac{p}{\nu_0}, \quad t_{11} = \frac{m}{\nu_1} \tag{2.20}$$

The process reaches equilibrium state if transitions between the states happen with the same probability:

$$p_{01} \, P_{0out} = p_{10} \, P_{1out} \tag{2.21}$$

With $P_{1out} = 1 - P_{0out}$ we get

$$P_{0out} = \frac{p_{10}}{p_{01} + p_{10}} \tag{2.22}$$

This is the probability that, if a spike is emitted by the network, it is emitted by neuron 0. If $m = p$, then $p_{01} = p_{11} = 1 - p_{10}$, so $P_{0out} = p_{10}$ is equal to the initial probability given by Equation 2.8.

The mean output inter-spike interval $\langle \Delta \rangle$ is given by the weighted sum of the transition times:

$$\langle \Delta \rangle = P_0 p_{00} \frac{m}{\nu_0} + P_1 p_{10} \frac{p}{\nu_0} + P_0 p_{01} \frac{p}{\nu_1} + P_1 p_{11} \frac{m}{\nu_1} \tag{2.23}$$

The mean output rate of the network is then:

$$\zeta = \frac{1}{\langle \Delta \rangle} = \frac{\nu_0 \nu_1 (p_{10} + p_{01})}{\nu_0 p_{01} (p_{10} p + p_{11} m) + \nu_1 p_{10} (p_{01} p + p_{00} m)} \tag{2.24}$$

2.2.1 Effect of self-excitation

We can now quantify the effect of the self-excitation on the performance of the network. We assume $V_I = V_{th}$ so p is independent of the membrane potential at the time of the inhibition (Equation 2.15). Strong self-excitation improves the probability that an output spike of the network is emitted from the neuron with the higher firing rate (Figure 2.7). The output rate for a network that integrates over many input spikes and has strong self-excitation is equal to a network that integrates over fewer input spikes and has no self-excitation.

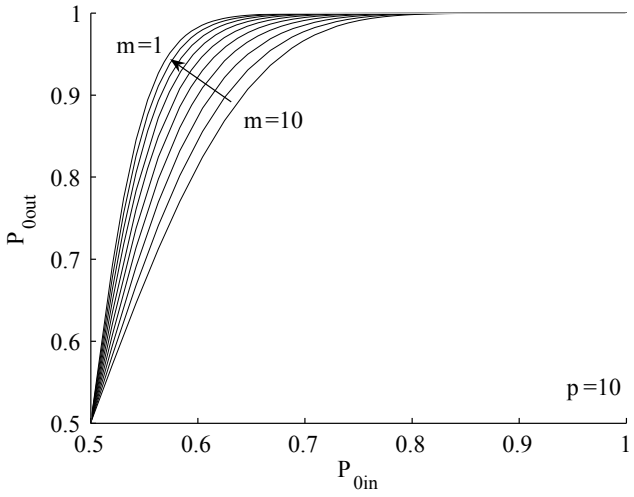

Figure 2.7: Effect of self-excitation: the probability of a correct decision P_{0out} increases as the self-excitation V_{self} is increased. m (p) is the number of spikes a neuron needs to reach threshold after self-excitation (global inhibition). For $m = p$ the curve is equal to the one shown in Figure 2.5 for $n = 10$. For $p = 10$ m is varied from $1 \ldots 10$. As the self-excitation is increased, the probability of a correct decision increases. Numerical evaluation of Equation 2.22.

2.2.2 Effect of inhibition

Until now, we assumed the inhibition to be strong, that is one inhibitory pulse discharges the membrane potential to the reset voltage. We now discuss the changes in performance if the inhibition is weakened.

In Equation 2.15 we assumed that the neuron that did not spike needs p spikes to reach threshold after being inhibited, while the neuron that spiked last needs m spikes. If fewer spikes are needed to reach threshold after inhibition (p) than after reset and self-excitation (m), a neuron that did not spike before will have a greater probability of spiking next. The neuron that did spike

before will have a lower probability of spiking next. In the initial state the neuron that receives the strongest input is the most probable one to spike. In the next step the network will then select a neuron that does to have spike before and the probability of a correct decision of the network decreases. Lowering the strength of the inhibition will therefore lead to a decrease in performance.

Our description does not make any predictions about the membrane voltage V_{E1}^- of the non-winning neuron before the output spike, except $0 < V_{E1}^- < V_{th}$ because the non-winning neuron did not spike. Let us assume that the membrane was sitting close to threshold before the neuron receives inhibition V_I. After inhibition $V_{E1}^+ = V_{th} - V_I$. The non-winning neuron will then reach threshold with $p \approx V_I/V_E$ spikes. With this assumption, weakening the inhibition leads to a drastic decrease in performance (Figure 2.8, left). But depending on the history of the spike input to the non-winning neuron, the membrane potential will be significantly lower than V_{th} before the inhibition, that is less inhibition is needed to achieve the same effect. We can address this in simulation (Figure 2.8, right), showing that the network performance does not decrease that rapidly. For $m = 10$, inhibition can be decreased to about 70% of V_{th} before the network shows a significant decrease in performance.

We conclude that weakening the strong inhibition always leads a decrease in performance. For weak inhibition, the functional description underestimates the performance of the network, while simulation shows that inhibition can be decreased to about 70% of V_{th} before the network shows a significant decrease in performance.

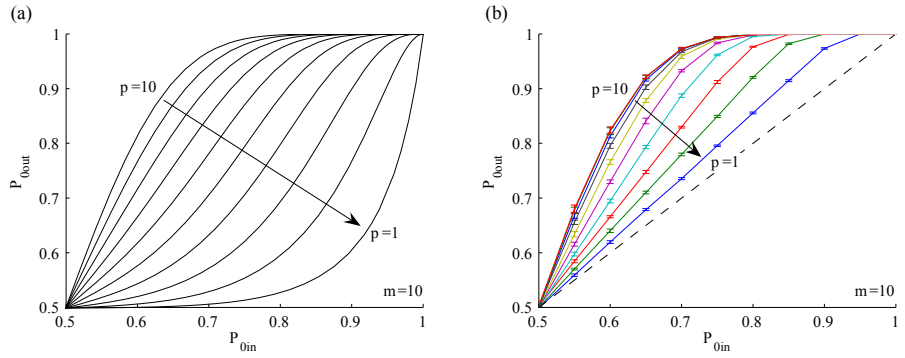

Figure 2.8: Effect of inhibition. Both graphs show the performance of the network if the strength of the inhibition is $V_I = p/nV_{th}$. In each graph, p is varied from 10 to 1 (top left to bottom curve, $m = 10$). Left: numerical evaluation of the functional description that assumes that the membrane potential of the non-winning neuron is close to threshold before inhibition. Right: simulation results. As can be seen by the simulation results, the assumption severely underestimates the performance of the network.

2.2.3 Generalization to Networks of N Neurons

We extend the simplified case of two neurons to a network with N neurons. Every neuron k receives a Poisson spike train of ν_k. The initial probabilities are

$$P_{init,k} = \int_0^\infty \nu_k \mathrm{P}(\nu_k t, n-1) \cdot \prod_{\substack{j=1 \\ j\neq k}}^{\mathrm{N}} \left(\sum_{i=0}^{n-1} \mathrm{P}(\nu_j t, i) \right) \mathrm{d}t \tag{2.25}$$

The transitions can be described by the matrices $(\mathbf{p}, \mathbf{t})$ with the transitions probabilities p_{kl} and transition times t_{kl} $(k, l \in 1...N)$:

$$p_{kk} = \int_0^\infty \nu_k \, \mathrm{P}(\nu_k t, m-1) \cdot \prod_{\substack{j=1 \\ j\neq k}}^{\mathrm{N}} \left(\sum_{i=0}^{p-1} \mathrm{P}(\nu_j t, i) \right) \mathrm{d}t \tag{2.26}$$

$$p_{kl} = \int_0^\infty \nu_l \, \mathrm{P}(\nu_l t, p-1) \cdot \sum_{i=0}^{m-1} \mathrm{P}(\nu_k t, i) \cdot \prod_{\substack{j=1 \\ j\neq k,l}}^{\mathrm{N}} \left(\sum_{i=0}^{p-1} \mathrm{P}(\nu_j t, i) \right) \mathrm{d}t \tag{2.27}$$

$$t_{kk} = \frac{m}{\nu_k}, \quad t_{kl} = \frac{p}{\nu_l} \tag{2.28}$$

The probability of an output spike of the network originating from a neuron k is then given by a vector $\mathbf{P}$. In the equilibrium state, $\mathbf{P}^*$ is the first eigenvector of matrix $\mathbf{p}$.

2.2.4 Interpretation in Terms of Information Theory

Let us discuss the classification of the winner-take-all network in terms of information theory, see Figure 2.9. We assume that the world consists of stimuli which the sensors and the pre-processing encode in spike trains. The task of the winner-take-all network is to classify which stimulus is present by observing the spike trains. What is the mutual information, the information that the observation of one or several spikes provides about the stimulus? How is the mutual information changed by the processing in the winner-take-all network?

The mutual information makes only predictions about the presence of features in the world. It does not quantify all the information about the world that is contained in the spike trains. The latter, the total amount of information that can be encoded in a spike train of certain length (the entropy of a spike train), is useful for a reconstruction of the world based on sensory output. [Reich et al., 2001] calls this the *formal* information, while the *attribute-specific* information asks how much information about a specific attribute is contained in the spike train.

In our model of the world, the stimulus X can take two values '0' and '1'. The stimulus is encoded into spike trains by sensor and pre-processing. As discussed in Figure 2.4, we assume that the winner-take-all receives a spike

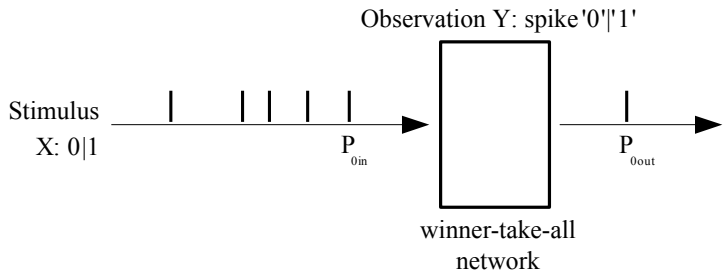

Figure 2.9: Interpretation of the winner-take-all network in terms of information theory. The stimulus X consists of the two symbols '0' and '1', which are encoded by sensor and pre-processing as spike trains. Every spike encodes either for symbol '0' or '1', with P_{0in} the probability of a spike coding for stimulus '0'. At the output of the winner-take-all the probability of a spike coding for stimulus '0' is P_{0out}. We discuss the mutual information, that is the average information about the stimulus X obtained by observing one input or output spike Y.

train in which the spikes encode either for symbol '0' or '1'. The probability of a spike coding for '0' is P_{0in}, of a spike coding for '1' is $(1-P_{0in})$. The output of the winner-take-all is a spike train with the same encoding, with the probability of a spike coding for '0' is P_{0out} and $(1 - P_{0out})$ for a spike coding for '1'.

P_{0in} depends on the quality of the sensor and the pre-processing, that is how good the stimulus is represented in the spike trains. Large variation, for example through noise, will result in a low P_{0in}, that is the spike train contains both spikes encoding for stimulus '0' and '1'. In the limit case of $P_{0in} = 0.5$, the spike trains would not depend on the stimulus at all.

P_{0out} depends on P_{0in} and on the processing in the winner-take-all network, that is on the number of spikes n the neurons need to reach threshold. For $n = 1$ the winner-take-all does not perform any computation, so we can regard an input spike as a special case of an output spike with $n = 1$:

$$P_{0in} = P_{0out}\Big|_{n=1} \tag{2.29}$$

The output Y is the observation of one of these output spikes. The average information that Y provides about the stimulus X is the mutual information defined as [Rieke et al., 1996, pp.122]:

$$I = \sum_X P(X) \sum_Y P(Y|X) \log_2 \frac{P(Y|X)}{P(Y)} \tag{2.30}$$

For simplicity, we assume that the stimulus takes the two values 0 and 1 with equal probability ($P(X = 0) = P(X = 1) = 0.5$). The entropy of X is then

$S(X) = 1\text{bit}$. The conditional probability $P(Y|X)$ for observing a spike at the input of the winner-take-all is given by the probability P_{0out} (respectively P_{0in} for $n = 1$):

$$\begin{aligned} P(Y=0|X=0) &= P(Y=1|X=1) = P_{0out} && (2.31)\\ P(Y=1|X=0) &= P(Y=0|X=1) = 1-P_{0out} && (2.32)\\ & && (2.33) \end{aligned}$$

Since these probability are symmetric, we get $P(Y=0)=P(Y=1)=0.5$:

$$P(Y) = \sum_X P(X,Y) = \sum_X P(X)P(Y|X) = P(X) \tag{2.34}$$

We can now rewrite the Equation 2.30 for our model, to get the mutual information about the stimulus from observing an output spike of the winner-take-all:

$$I = P_{0out}\log_2(2P_{0out}) + (1-P_{0out})\log_2(2(1-P_{0out})) \tag{2.35}$$

P_{0out} and the mutual information depend on the quality of sensor and preprocessing (P_{0in}) and number of spikes n the winner-take-all neurons need to reach threshold. Figure 2.10 shows this dependence.

The maximal information that can be gained by observing an spike is the entropy of the stimulus $S(X)=1\text{bit}$ itself.

Our winner-take-all model is equivalent to counting the input spikes. Observing one output spike of the winner-take-all is equivalent to observing the input spike train during the time the winner-take-all is integrating.

The winner-take-all network therefore does not change the information transmitted per time, but it increases the information per spike, transforming the spike train into a sparse representation. Encoding the same information with transmitting less symbols is equivalent to compression. The compression factor is the ratio of input spike rate of the network over the output spike rate

$$f = \frac{\nu}{\zeta} \tag{2.36}$$

with ν the input spike rate and ζ the output spike rate defined Equation 2.13.

2.3 Time-Varying Firing Rates

We now extend our analysis from stationary inputs to inputs with time-varying firing rates.

It is a fundamental principle of the Poisson distribution that there are no assumptions on when events occur in a certain time interval, but only on the number of spikes in the total interval. At any given time, the spike rate is given

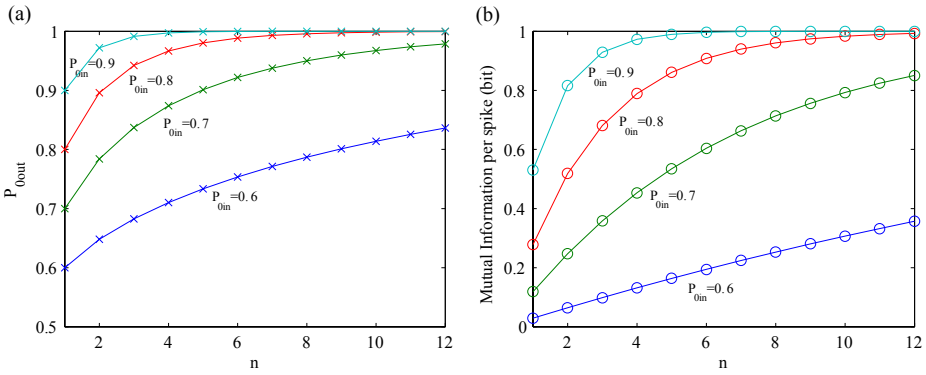

Figure 2.10: (a) Probability that an output spike of the winner-take-all encodes for the correct stimulus, versus the number of input spikes the neurons need to reach threshold n. For $n = 1$ the winner-take-all does not perform any computation and the output probability is identical to the input. Four input probabilities (0.6,0.7,0.8,0.9) are shown. (b) Mutual information, the average information that the observation of one output spike provides about the stimulus, dependent on the number of input spikes the neurons need to reach threshold n. For $n = 1$ the winner-take-all does not perform any computation, and the mutual information is equivalent to that obtained by observing one input spike. With increasing number the neurons of the network are integrating over the mutual information converges to the stimulus entropy which is 1bit. The same input probabilities as in (a) are shown. Observing an output spike of the winner-take-all is equivalent to counting the same number of input spikes. The winner-take-all transforms the input spike train into a more sparse representation, by increasing the mutual information *per spike*.

by the probability of a spike occurring. We can therefore replace the number of spikes νt in formula 2.8 with the integral $\int_0^t \nu(t')\,\mathrm{d}t'$. This is the area under the input spike rate function over time. Equation 2.25 generalizes to

$$P_0 = \int_0^\infty \mathrm{P}(\int_0^t \nu_k(t')\,\mathrm{d}t', n-1) \cdot \nu_k(t) \cdot \prod_{\substack{j=0 \\ j\neq k}}^{N} \left(\sum_{i=0}^{n-1} \mathrm{P}(\int_0^t \nu_j(t')\,\mathrm{d}t', i) \right) \mathrm{d}t \quad (2.37)$$

2.3.1 Switching of Stronger Input

We first discuss the case that the stronger input switches from one neuron to the other. With self-excitation, the network shows hysteretic behavior. As shown in Figure 2.7, self-excitation can result in the same performance as integrating over fewer spikes but without self-excitation. To quantify this effect for time-variant input, we discuss the receiver operating characteristics (ROC) of the network.

We assume that neuron '1' receives the stronger input before switching occurs at time $t = 0$ and that this input was applied long enough so that the network reached a stable state so that the output also encodes for neuron '1'. From time $t = 0$ on, neuron 0 receives the stronger input.

We can calculate how many subsequent spikes are emitted by neuron '1' before the neuron '0' will spike. The probability of having exactly k subsequent spikes from neuron 1 is the probability of staying k times in state 1, and then make a transition to state 0:

$$(p_{11})^k \cdot p_{10} \quad (2.38)$$

Summing over all possible k results in the average number of spikes k_1 that neuron 1 emits before neuron 0 spikes:

$$k_1 = p_{10} \sum_{k=1}^{\infty} k\,(p_{11})^k = p_{10}\,\frac{p_{11}}{(1-p_{11})^2} = \frac{p_{11}}{p_{10}} \quad (2.39)$$

For every output spike, neuron '1' integrates m input spikes that arrive with rate ν_1. At the moment when the switch from neuron '1' to neuron '0' occurs, neuron '0' has to integrate additional p input spikes with rate ν_0. The average time to switch from neuron '1' to neuron '0' is then:

$$t_{01} = \frac{p_{11}}{p_{10}}\,\frac{m}{\nu_1} + \frac{p}{\nu_0} \quad (2.40)$$

The average switching time increases with the self-excitation. t_{01} is the time during which the network emits spikes from the neuron that does not receive the larger spike rate, if this neuron spiked by chance or if the stronger input changed target from neuron '1' to neuron '0'. In this time, neuron '1' will emit k_1 transient spikes.

We consider the first spike of neuron 0 as the criterion for a correct detection of the changed inputs. We quantify this decision with two variables:

$P_{TP}(t)$ the probability that neuron '0' spikes in time t after the inputs are switched, that is the probability that the network indicates a correct decision (*true positive*).

$P_{FP}(t)$ the probability that neuron '0' spikes in time t if the rates did not switch, that is the probability that the network indicates a switch even though the rates did not change (*false positive*).

With $\nu_0 > \nu_1$ after the switch, the fraction of a true positive after time t can be calculated from the probability that no switch is detected and only neuron '1' is firing. In time t, neuron '1' receives on average $\nu_1 t$ input spikes and makes on average $\nu_1 t/m$ output spikes. This is equivalent to the case the network stays $\nu_1 t/m$ times in state 1:

$$P_{TP}(t) = 1 - (p_{11})^{\frac{t\nu_1}{m}} \tag{2.41}$$

A false positive is a state transition from state 0 to state 1, if we keep the convention that $\nu_0 > \nu_1$. We model this case again by the probability of the contrary event that no transition occurs from the neuron with the stronger input (neuron '0'):

$$P_{FP}(t) = 1 - (p_{00})^{\frac{t\nu_0}{m}} \tag{2.42}$$

Figure 2.11 shows the true and false positive rates similar to the receiver operating characteristic curve (ROC). On ROC curves the performance of the classifier is shown in dependence of a parameter. Varying this parameter determines the true and false positive probabilities of the classifier. For our classifier, we use time as the parameter and draw true and false positive probabilities dependent on the time when the network spikes. As for the ROC curve, the area under the curve quantifies the classifier performance. We follow the engineering definition of the area under the curve, which measures the area between curve and chance level (dashed line). We call this the discrimination performance. In Figure 2.12 we compare the discrimination performance in dependence of the average switching time, by varying the amount of self-excitation. The discrimination performance is higher if no self-excitation and a longer integration time is used compared to a shorter integration time with self-excitation.

We can conclude that for switching input, self-excitation does not lead to better performance: better discrimination performance is obtained if the neurons integrate over more input spikes than if they receive self-excitation, for the same hysteresis of the network. This is consistent with our view of self-excitation in the contect information that we discuss in Section 2.5: self-excitation helps to exploit the fact that the inputs are stationary, which is not the case for switching input.

2.3.2 Moving Input

One can think of an infinite number of scenarios that involve stimuli with non-stationary spike rates. We will discuss an – as we think – representative example

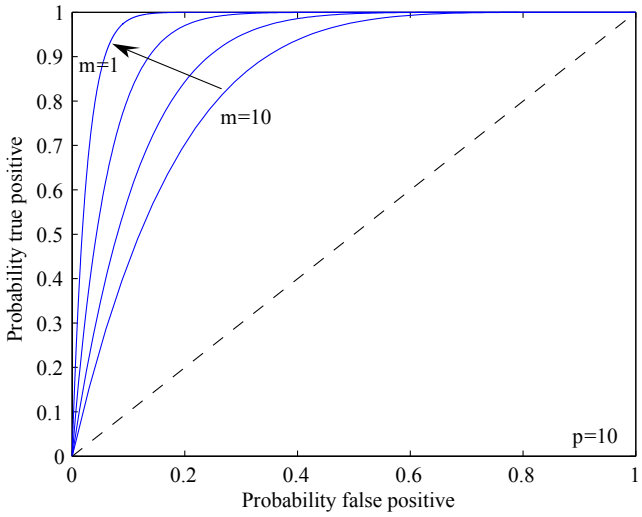

Figure 2.11: Classifier characteristics: probability of true positives versus false positives, similar to the common receiver-operating-characteristics (ROC) curves. Different amounts of self-excitation are shown. The network has to detect a switch in the neuron that receives the stronger input. The stronger input has a spike rate 50% higher than the other input. The curves show the classification performance dependent on the time when the network spikes first after the switch. The area between curve and chance level (dashed line) quantifies the classifier performance, the discrimination performance.

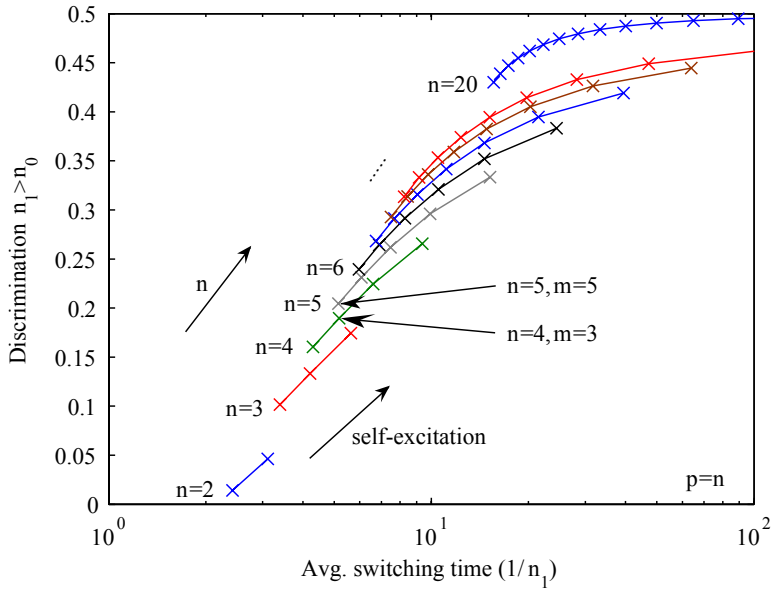

Figure 2.12: Classifier characteristics: discrimination performance versus the average switching time, the hysteresis of the network. The discrimination performance is obtained from Figure 2.11. Every curve corresponds to the number of spikes the neurons need to reach threshold (n). Within each curve the self-excitation is increased: at the left-most data point of each curve there is no self-excitation ($m = n$), then for each data point along the curve the self-excitation is increased by one, that means the winning neuron needs one spike less to reach threshold ($m = n \ldots 1$). The discrimination performance is better for more spikes to reach threshold and less self-excitation than for stronger self-excitation, see the indicated points: at the point ($n = 4, m = 3$) the neurons need 4 spikes to reach threshold and receive self-excitation equivalent to one spike. At the point ($n = 5, m = 5$) the neurons need 5 spikes to reach threshold and do not receive self-excitation. At ($n = 5, m = 5$) the network has a higher discrimination performance at about the same average switching time. Similar comparison can be made for any other pair of data points of (no self-excitation/higher n) versus (self-excitation/smaller n).

here: a Gauss-shape spike wave that is traveling across a line of winner-take-all neurons. The Gauss shape is a good generalization of the output of an earlier pre-processing stage, since it models the output of a matched-filter convolution operation with a reasonable complex kernel. In our case, we assume a receptive-field-like pre-processing that is repeated for each neuron. The input then results from an object that is moving along the receptive fields. Near the 'operating point', that is around the neuron that we will analyze, we consider the speed of the object to be constant. This example case is simple enough to be functionally analyzed, but captures also the more complex case of the CAVIAR data, as we will show in Chapter 4. An overview of the model is shown in Figure 2.13.

The neurons are arranged in one infinite line of neurons with numbers $i = -\infty \ldots \infty$. Without loss of generality, our analysis regards neuron 0. The temporal distance between the neurons is d. The time-dependent input to a neuron i, $\nu_i(t)$, is a Gaussian function centered on that neuron:

$$\nu_i(t) = \frac{\nu_{max}}{2\pi\sigma^2} \exp\left(-\frac{(t-di)^2}{2\sigma^2}\right) \tag{2.43}$$

The spike rate is normalized so that the neuron receives the rate ν_{max} when the input wave is aligned to the neuron.

We can compare the temporal distance d to the discriminability measure $\hat{d}$ used in psychophysics: It measures the distance between the center of two Gaussian distributions that are normalized to $\sigma = 1$. $\hat{d}$ is the overlap between the distributions, which reflects the difficulty in separating the two distributions. In our example, d is the distance between two neurons that receive spike input consecutively. Nevertheless, d is also a measure of the difficulty in classifying the ball position like the discriminibility measure in psychophysics.

As discussed in Section 2.3.1, the performance of the winner-take-all is best if the output is as sparse as possible. In this example, each neuron should make on average one spike when the object passes over it. The average integration time of the winner-take-all decision is then the time the object needs to move from one neuron to the next, d. It is natural to center this interval on the neuron, for neuron 0 at $t = 0$. In this time the neuron receives a higher firing than all its neighbors (see Figure 2.13). Integration of inputs to the winner-take-all starts then at the intersection of two neighboring input firing rates at $t = -d/2$. We use the variable T to indicate the integration time ($T = t + d/2$). To start the analysis we assume that all neurons are reset at this time.

We define a helper function for the area under the input spike rate function for each neuron i in the integration time T:

$$\mathrm{R}_i(T) = \int_{-d/2}^{T-d/2} \nu_i(t)\,\mathrm{d}t = \frac{\nu_{max}}{2\pi\sigma^2} \int_{-d/2}^{T-d/2} \exp\left(-\frac{(t-di)^2}{2\sigma^2}\right)\,\mathrm{d}t \tag{2.44}$$

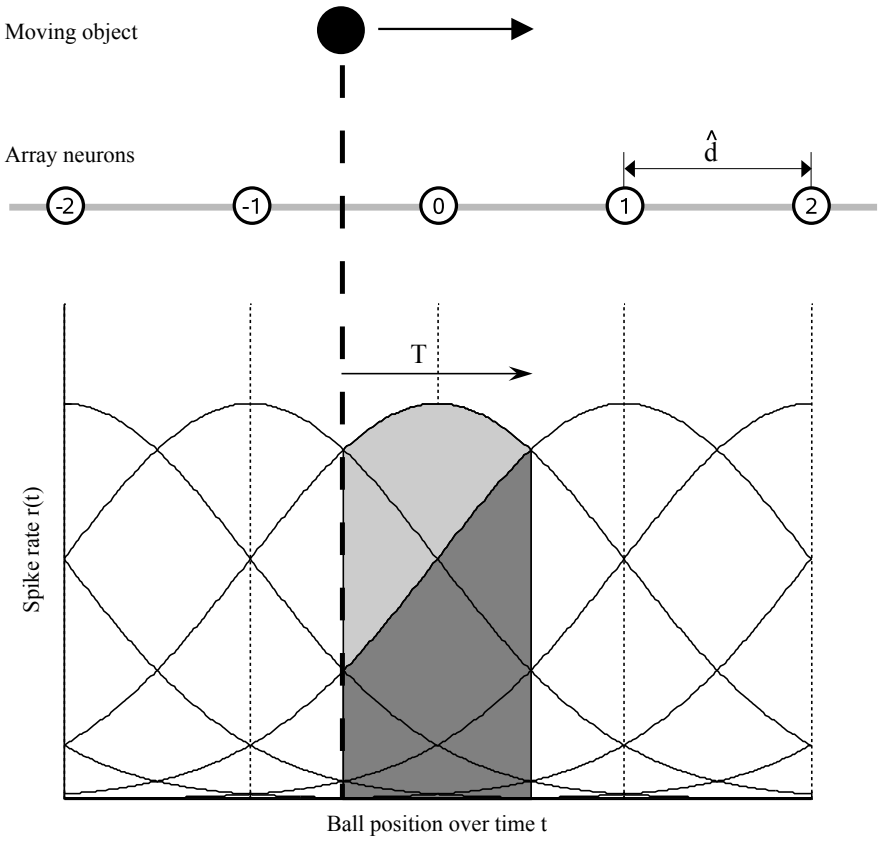

Figure 2.13: Model of a moving input: An object moves across a line of neurons of the winner-take-all network, producing a wave of activity. The neurons are equidistantly spaced: assuming a constant velocity of the object, this results in the temporal distance d. The neurons are numbered from $-\infty$ to ∞. The diagram shows the spike rate evolving over time. At a certain point in time (thick dashed line), neuron 0 receives an input spike rate indicated by the curve that peaks at its position. Integration of the winner-take-all network starts at the intersection of two neighboring input curves. At an integration time T the neuron 0 aligned to the ball position receives a number of spikes equivalent to the light gray area under its spike input curve, whereas a neighboring neuron receives a number of spikes equivalent to the dark gray area.

On average, the winner should spike at $T = d$, receiving n spikes:

$$n = \mathrm{R}_0(d) = \frac{\nu_{max}}{2\pi\sigma^2} \int_{-d/2}^{d/2} \exp\left(-\frac{t^2}{2\sigma^2}\right) \mathrm{d}t \tag{2.45}$$

We will refer most of our analysis to n, the number of spikes the neurons need to reach threshold, instead of r_{max}, to be able to make comparisons to Section 2.1.

We rewrite Equation 2.37 for the line of neurons $i = -\infty \ldots \infty$ and the area under the input spike rate function $\mathrm{R}_i(T)$:

$$P_{n,d,\sigma} = \int_0^{\infty} \mathrm{P}\left(\mathrm{R}_0(T), n-1\right) \cdot \nu_0(t) \prod_{\substack{k=-\infty \\ k\neq 0}}^{+\infty} \left(\sum_{i=0}^{n-1} \mathrm{P}(\mathrm{R}_k(T), i)\right) \mathrm{d}T \tag{2.46}$$

This is the probability that neuron 0 makes the first output spike after a reset at time $t = -d/2$.

The rate of neuron 0 making the first output spike of the network after $d/2$ is equivalent to the rate of a correct decision:

$$r_{correct}(T) = \mathrm{P}\left(\mathrm{R}_0(T, n-1)\right) \cdot \nu_0(T - d/2) \cdot \prod_{\substack{k=-\infty \\ k\neq 0}}^{+\infty} \left(\sum_{i=0}^{n-1} \mathrm{P}(\mathrm{R}_k(T), i)\right) \tag{2.47}$$

The rate of an incorrect decision of the network is:

$$r_{false}(T) = \sum_{\substack{j=-\infty \\ j\neq 0}}^{+\infty} \mathrm{P}\left(\mathrm{R}_j(T, n-1)\right) \cdot \nu_j(T - d/2) \cdot \prod_{\substack{k=-\infty \\ k\neq j}}^{+\infty} \left(\sum_{i=0}^{n-1} \mathrm{P}(\mathrm{R}_k(T), i)\right) \tag{2.48}$$

How can we infer the ability of the network to reconstruct the position of the stimulus from the probability of a correct output spike and the output rate? The network discretizes the stimulus position in time and space. In time, because the output of the network is event(spike)-based; in space, since the network can report the stimulus position only at every neuron. The ideal result of this discretization is a staircase function as shown in Figure 2.14. This is equivalent to decoding the output of the network with a memory of one variable, the position, which is updated at every spike. Of course, one could think of a more elaborate processing of the network output such as spatial and temporal interpolation. But this interpolation is limited by the accuracy of the network output, which is exactly what we analyse here.

Deviation of the stimulus position reconstructed from network output from the ideal case results from two types of errors: jitter in the timing of the output spikes; and spikes from neurons that are not aligned to the stimulus position. We will summarize the effect of both by defining the total position error e as the area between the stimulus position reconstructed from the network output and

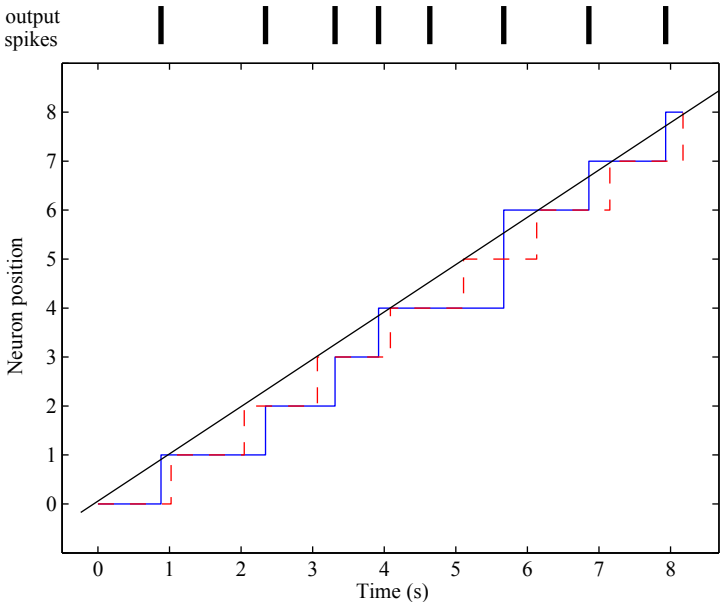

Figure 2.14: Reconstruction of object position from winner-take-all output. The stimulus travels along a line of neurons with constant velocity (straight line). The winner-take-all network reports the object position by eliciting a spike from the neuron that the object is aligned to. The ideal output is a staircase function that discretizes the object position to the neuron number (dashed line). The network emits spikes indicating position updates (shown above the plot). With Poisson input, the object position can not be determined optimally (continuous staircase function). The output spikes are shifted earlier or later (jitter error); or the networks fails to detect the correct position (classification error), such as at position 5. We define the total position error as the area between ideal and reconstructed stimulus position (area between dashed and continuous staircase function), normalized to the number of neurons and the temporal distance d. Simulation data ($n = 10, d = 1, \sigma = 1$).

the ideal case (see Figure 2.14). We norm this area to the number of neurons, so that e denotes an average deviation of the reconstructed stimulus position per neuron.

The jitter error is caused by variation of the time when the first neuron of the network reaches threshold. We define e_{jitter} as the area between the time of the first output spike of the network and the average time d, normalized to d:

$$e_{jitter} = \frac{1}{d} \int_{-\infty}^{+\infty} |\, T - d \,| \; (r_{correct}(T) + r_{false}(T)) \; \mathrm{d}T \tag{2.49}$$

Spikes from neurons that are not aligned to the stimulus position contribute to the classification error e_{class}. We assume that if the network decides on an incorrect stimulus position, this position will be kept until the next spike of the network at time d later. We again define the error as the area between the correct position 0 and j, weighted by the probability P_j that neuron j makes the first output spike of the network.

$$\begin{aligned} e_{class} &= \sum_{j=-\infty}^{+\infty} j \; P_j \qquad (2.50) \\ &= \sum_{j=-\infty}^{+\infty} j \int_{0}^{\infty} \mathrm{P}\left(\mathrm{R}_j(T), n-1\right) \cdot \nu_j(t) \cdot \prod_{\substack{k=-\infty \\ k \neq j}}^{+\infty} \left(\sum_{i=0}^{n-1} \mathrm{P}(\mathrm{R}_k(T), i) \right) \mathrm{d}T \end{aligned}$$

We can add these two errors together to obtain the total position error e as defined before.

$$e \approx e_{jitter} + e_{class} \tag{2.51}$$

However, there are two problems in this approximation: the jitter area and the classification error overlap, which means that the same area is accounted for twice. This is dominant when both jitter and classification errors are large for small n. The second problem is our assumption of the start of the integration at the intersection of the input spike rate functions of two neighboring neurons ($t = -d/2$). Due to the jitter in the first output spike, the next integration period will actually start earlier or later. Results of the position error obtained by evaluation of the functional description can therefore not be directly compared to simulation data. Nevertheless, the data fits qualitatively (Figure 2.15).

2.4 Dynamic Synapses

Most synapses in cortex have temporal dynamics, that is they show depressing and facilitating dynamics, see for example [Markram et al., 1998]. Depressing synapses decrease their synaptic efficacy if stimulated with a continuous train of action potentials. Facilitating synapses transiently increase their efficacy

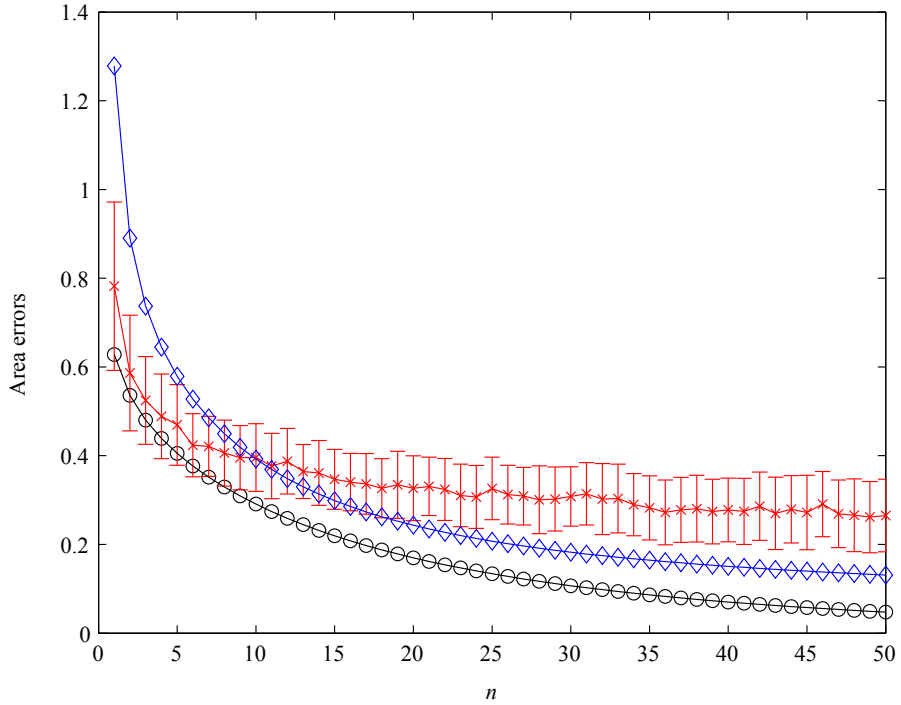

Figure 2.15: Development of area errors dependent on the number of spikes n to reach threshold. Shown are the jitter error (variation of the timing of the output spikes, diamonds) and the classification error (output spikes from neurons not aligned to the stimulus position, circles), from numerical evaluations of Eqns. 2.49 and 2.51. For comparison, the total area error is shown from simulation results (crosses). In simulations, the complete line of neurons is processed, so that the start of the integration period of the winner-take-all depends on the time of the last output spike, whereas in the functional description the integration always starts at the optimal intersection of two neighboring input waveforms. For low n, jitter and classification error include the same area twice, therefore the sum of both would exceed the error obtained in the simulation results. Error bars of the simulation data denote one standard deviation ($d = 1, \sigma = 1$, every data point result of 100 trials with 20 neurons).

and undergo a depression phase afterwards. The types of behavior and their parameters found in cortex show a large variety.

A model of dynamic synapses defined in [Markram et al., 1998] is formulated with four parameters: absolute synaptic efficacy (A), utilization of synaptic efficacy (U), recovery from depression (τ_{rec}) and facilitating time constant (τ_{fac}). At each action potential, a fraction u of the available synaptic efficacy R is used. This fraction is instantaneously unavailable and recovers with a time constant τ_{rec}. The facilitation mechanism increases the fraction u used at each action potential, starting with U and decaying with the time constant τ_{fac}. The parameters are described by the equations

$$R_{n+1} = 1 + (R_n(1 - u_{n+1}) - 1) \exp\left(\frac{-\Delta t}{\tau_{rec}}\right) \tag{2.52}$$

$$u_{n+1} = U + u_n(1 - U) \exp\left(\frac{-\Delta t}{\tau_{fac}}\right) \tag{2.53}$$

Δt is the inter-spike-interval between the nth and the $(n+1)$th action potential. The synaptic response generated by the nth action potential is

$$w_n = A \cdot R_n \cdot u_n \tag{2.54}$$

A is a scaling factor to adjust the absolute synaptic efficacy.

If only depression is used, τ_{fac} is set to zero and hence $u_n = U$. [Abbott et al., 1997] formulates Equation 2.52 with $f = 1 - U$ in the depressing case:

$$R_n = 1 + (fR_n - 1) \exp\left(\frac{-\Delta t}{\tau_{rec}}\right) \tag{2.55}$$

The response of the facilitating synapse to a step function in the input is shown in Figure 2.16. Facilitation induces a short transient increase, before the synapse goes into the depressed state.

For the facilitating synapse, we use $U = 0.05$, $\tau_{rec} = 0.4s$ and $\tau_{fac} = 1.8s$ as reported as mean values in [Markram et al., 1998]. For the depressing synapse we started our simulations with $f = 0.7$ and $\tau_{rec} = 0.2s$ as reported in [Abbott et al., 1997]. In both references the variation of the parameters is above 50%.

Obviously, the steady-state response of facilitating and depressing synapses is much smaller than their peak transient response and they will therefore suppress any stationary input like background noise. But what is the effect on the input signal itself, if we assume that it has the same profile as described in Section 2.3.2?

To compare the outputs, we weighted the input spikes with the synaptic efficacy at an input spike. Averaging over many trials results in the product of the mean spike rate and the mean synaptic efficacy over time, $r(t) \cdot w(t)$. [Abbott et al., 1997, Markram et al., 1998] define the synaptic efficacy as the jump in the membrane potential of the post-synaptic potential, the EPSP. The product

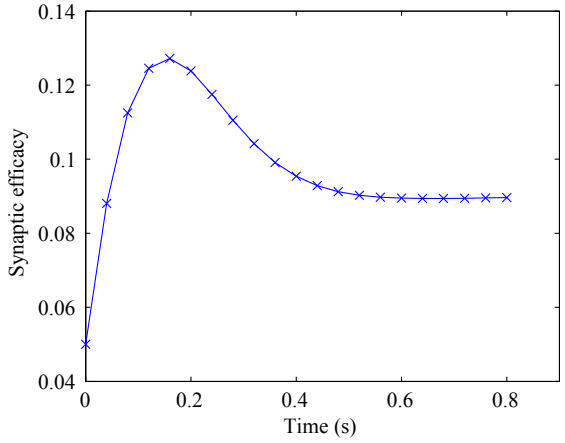

Figure 2.16: Facilitating Synapse. Development of synaptic efficacy in response to an input step function of a regular rate of 30Hz, starting at $t = 0$. The facilitating synapse was implemented after [Markram et al., 1998], with U=0.05, τ_{rec}=0.4s and τ_{facil}=1.8s.

has the units V/s and the total input to the postsynaptic neuron is the integral $\int r(t)w(t)\,dt$. In the case of non-dynamic synapses, input spikes arrive with a Poisson distribution of a certain rate and each spike has a synaptic efficacy of one. To have the same effect in the depressing (facilitating) case, that is the same area under the curve $r(t)w(t)$, the synaptic efficacies are less (greater) than 1 and the spike rate is higher (lower) than in the non-dynamic case. We can then apply our model by replacing $r(t)$ with $r(t)\cdot w(t)$, although the statistical properties of the input currents are different.

Depressing synapses broaden the input wave form, depending on the time constant of the synapse (Figure 2.17). For very short time constants, the synapse recovers before the next spike and the waveform is not significantly changed. For shorter and intermediate time constants, the waveform is broadened since the peak is reduced as the synapse goes into the depressed state. For longer time constants, the synapse acts as an onset detector, resulting in a waveform with a short peak and a tail with low or very low firing rate. The neuron is stimulated only at the start of the input. The synapse is sensitive to changes in input, for example fluctuations in background noise [Abbott et al., 1997] and will therefore not improve the signal quality. In the case of weak depression, broadening the input signal is similar to decreasing the discriminability $d = 1, \sigma = 1$ in our model, leading to decreased classification performance.

For a facilitating synapse, we expect a sharpening of the input. If the time constants of the synapse are adjusted to match that of the input waveform, the synapse is in the facilitating state during the rising edge of the input waveform

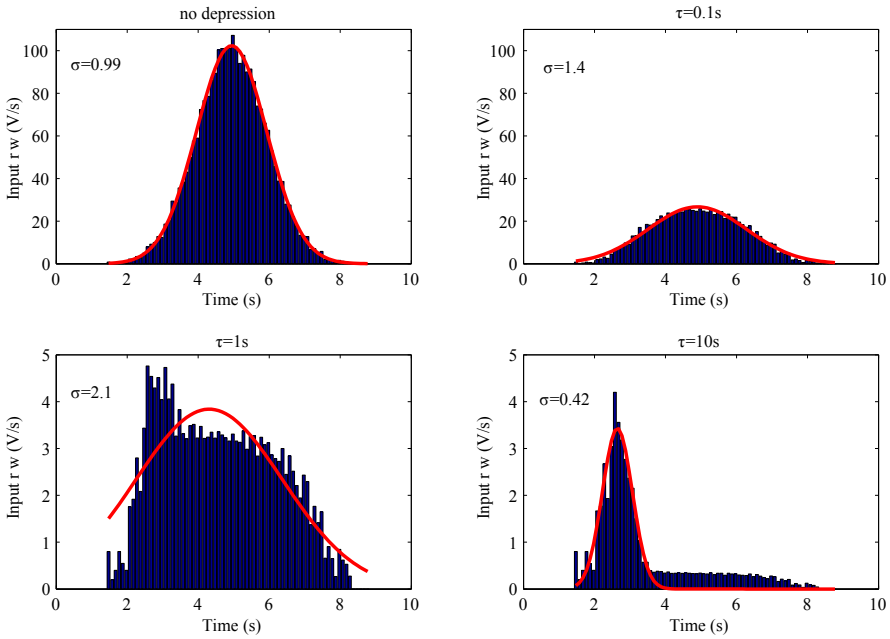

Figure 2.17: Effect of depressing synapses on the input waveform. The input spike train is a Gaussian wave form with a peak spike rate of 100Hz and a standard deviation of $\sigma = 1$. To illustrate the effect of the synaptic dynamics, each input spike is weighted by the synaptic efficacy at the arrival time of the spike (w). The resulting product $(r \cdot w)$ is the average input to the soma. The synaptic efficacy is decreased by 30% for each input spike and recovers with a time constant of 0.2s. Each graph shows the average of 50 trials. Without depression, the synaptic efficacy is constant (we assume w=1) and the waveform of the product of spike rate and synaptic efficacy is equivalent to that of the spike rate itself (top left). For weak synaptic depression, that is fast recovery of the depression, the input wave form is broadened and its peak amplitude decreased (top right). With stronger depression, that is slower recovery, the input wave is further broadened with the onset emphasized (bottom left). Even stronger depression leads to a transient response to the onset of the input, but fails to follow the input dynamics since the synapses are fully depressed at this point (bottom right).

and is depressed during the falling edge. Since the facilitating synapse model has four parameters and the input waveform only two (standard deviation and peak spike rate), we kept the parameters of the synapse constant as given in the literature [Markram et al., 1998], and adjusted the parameters of the wave form. We noticed that a certain minimum spike rate is necessary to achieve an observable effect, because otherwise the Poisson input does not drive the synapse into depression or facilitation consistently due to its variation.

We then varied the standard deviation of the Gaussian input (Figure 2.18). We can distinguish three cases. (1) The standard deviation of the input waveform is short compared to the time constants of the synapse, (2) the synapse is in the facilitating phase during the complete duration of the stimulation, and (3) the resulting input waveform is not significantly changed. If the standard deviation is too large, the synapse is mainly depressed during stimulation and acts as an onset detector, comparable to the synapse with only depression. If the synapse parameters match the input waveform, the synapses facilitates during the rising edge of the stimulus and depresses during the falling edge. This leads to a sharpening in the response of about 20%.

In conclusion, the effect of the dynamic synapses in our example of Gaussian distributed input is surprisingly weak: since the transient input sensor does not output any sustained responses (at least not strong enough to produce output of the convolution chip), depressing synapses are not necessary to suppress steady-state input.

Dynamic synapses roughly compute the derivative of the input. Computing the derivative of a Gaussian waveform results again in Gaussian shape, and that is the case why our input waveform does not change significantly for a wide range of synaptic parameters. Even with careful tuning of the synaptic parameters, the input to the neurons is only sharpened by 20%. The tuning range is in good agreement with studies of the coding of temporal information in dynamic synapses, in which information transfer is maximized near an optimal frequency [Fuhrmann et al., 2002].

We assumed that the neuron receives its total input through only one synapse. In a more biologically detailed model, the total input would be provided by inputs at several synapses. The effect of facilitation and depression would be even less enhanced, at least for the stimulation rates considered here.

We can imagine that dynamic synapses are effective in a different context, since they are widely evident in the cortex. If the input is a series of step functions, facilitating synapses transform each onset into a short burst. The resulting traveling series of bursts is similar to the series of Gaussian waveforms for which we showed that the performance of the winner-take-all can be close to optimal.

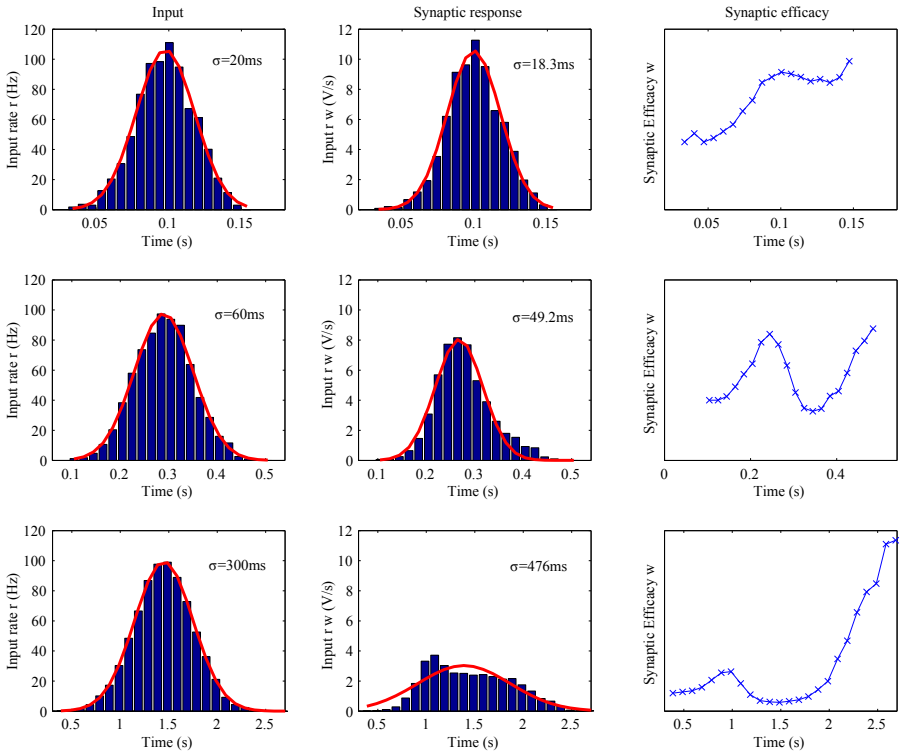

Figure 2.18: Effect of facilitating synapses on the input waveform. Input spike train is a Gaussian wave with a peak spike rate of 100Hz and different standard deviations (left column).Each input spike is weighted by the actual efficacy the synapse had at the arrival time of the spike (middle column). The synaptic efficacy is shown on the right column. The parameters of the facilitating synapse are taken from [Markram et al., 1998], except the absolute amplitude that only serves as a scaling factor. The time constants are $\tau_{rec} = 0.4s$ for the depression and $\tau_{fac} = 1.8s$ for facilitation. We omitted adjusting the absolute amplitude since we are only interested in the shape of the input waveform and indicated this by leaving the y-axis without units. Each graph shows the average of 250 trials. *(continued on next page)*

Figure 2.18 (*continued from last page*): We show three cases of example input: for short pulses, the facilitating synapse is still in the facilitating phase when the stimulation ends (top row). The resulting input waveform is therefore not significantly changed. If the synapse parameters match to the input waveform (middle row), the synapses facilitates during the rise of the stimulus and depresses during decrease, with the peak aligned at or shortly before the peak of the input. The input waveform is sharpened by about 20%. At the very end, the synapse leaves the depressed state since it is not driven hard enough any more. Due to the low input rate the strong synaptic efficacy does not contribute significantly during this time. For longer input pulses, the synapse indicates the onset and then broadens the input waveform in the depressing phase (bottom row). In this mode, the synapse is sensitive to any change in its input.

2.5 Context Information

As context information we summarize all computation that extends the basic knowledge which of the input channel is currently stronger. The context can be temporal, that is memory of the past winning neurons, or spatial, if the network structure encodes additional information. In this section we will discuss such concepts in a less strict manner than the analysis of the winner-take-all mechanism, to give the reader an outlook how the winner-take-all concept could be extended. The ideas presented here should be seen as starting points with preliminary data, not complete analyses.

We can interpret the self-excitation described in Section 2.2.1 as context information: if the input is stationary, the network can exploit the knowledge that, once the correct winning neuron is selected, this winner will continue to be the winner. Self-excitation provides this mechanism since it increases the probability that the neuron that spiked before will also spike next. Since the neuron with the strongest input has the highest probability to be selected as the winner, facilitating this neuron for the next output increases the accuracy of the network (Figure 2.7).

For moving stimuli as defined in Section 2.3.2, the network could exploit the fact that the object does not jump in space, but moves smoothly to neighboring locations of its last detected position. Instead of self-excitation, neighboring neurons are facilitated to increase their probability of spiking next. The width of the facilitation should reflect the traveling distance of the object from one output spike to the next. Facilitation of neighboring neurons can be implemented with lateral excitatory connections, resembling the strong lateral connectivity found in cortical circuits [Douglas and Martin, 2004]. We started to explore this scheme with neighboring excitation in the second chip implementation (for detail on the chip see Section 3.2). However, the mismatch in the used excitatory synapse made it difficult to estimate the increase in performance. Furthermore, the connectivity is unspecific. The connectivity is hardwired to excite all neigh-

boring neurons, whereas an object usually moves in one direction. [Woergoetter et al., 1999] describes a mechanism that propagates the movement information to neighboring neurons, but at the cost of introducing a rather unusual neuron model that requires special implementation.

2.5.1 Permitted Sets

Extending the local neighboring connectivity to embed structure in the network is formulated in the idea of permitted sets [Hahnloser et al., 2000]. A permitted set is a group of neurons that can be active at the same time. A forbidden set is any set of neurons in which the neurons are not allowed to be active at the same time. Permitted and forbidden sets can be implemented in two ways: (a) starting with a full mutual inhibitory connectivity, for each permitted set the inhibitory connections between all pairs of neurons of the set is removed [Xie et al., 2001, 2002]. (b) For each permitted set, mutual excitatory connections between all pairs of neurons of the set are added. The excitation is balanced by unspecific global inhibition [Hahnloser et al., 2003]. Detailed mathematical proofs based on the connectivity matrix can be summarized as follows: (1) every subset of a permitted set is also permitted, and (2) any superset of forbidden sets forms again a forbidden set. Following from (1) permitted sets can otherwise not overlap; they form one large set and loose specificity. The hard winner-take-all network is a special case of such networks in which all neurons form a forbidden set and no permitted sets with more than one neuron exists.

Excitation in permitted sets (model b) implements signal restoration which we discussed as part of the properties of soft winner-take-all circuits in Section 1. If the input activates only part of a permitted set, the excitation from these neurons will stimulate the remaining neurons of the permitted set and complete the pattern encoded in the connectivity. This behavior can also be found in the model (a), if the inhibitory connections between the permitted set neurons are not removed, but replaced by weak excitatory ones.

Adding excitatory connections on top of an unspecific inhibition (model b) is particularly attractive since it seems biologically more plausible. Since inhibition is always mediated by inhibitory neurons, in model (a) each permitted set would be represented by an inhibitory cell. This cell receives excitation from the neurons in its set and inhibits all neurons in the network, except the ones of its set. While still being a topic of discussion, such detailed local inhibitory connectivity does not seem plausible. In contrast, model (b) could be implemented by having unspecific local inhibition and long range excitation which seems to implement specific connectivities.

In biology inhibitory cells is always mediated by inhibitory cells and is therefore unspecific, while in the first model inhibition is very specific and would require a large number of specific inhibitory cells. However, the model (b) requires a very fine-tuned balance of excitation and inhibition in the network, since the excitation in the permitted set has to compensate the global inhibition. While

it is an open discussion if such an equilibrium is biologically plausible, it is difficult to implement with spiking neurons. The original model was formulated with linear threshold units [Hahnloser et al., 2003] or current-mode VLSI circuits [Hahnloser et al., 2000]. In spiking neurons, graded analog values can still be represented as firing rates if the variation in the spike times is limited. We explored an implementation in the second chip version, and it turned out to be difficult to establish a fine-tuned equilibrium between excitation and inhibition due to the mismatch on that chip version. We therefore concentrated on the model (a) with specific inhibition.

Figure 2.19 shows the principle of permitted sets with specific inhibitory connectivity. From a full mutual inhibitory connectivity, all connections between the neurons of each permitted set are removed. With input, these neurons evolve to a co-spiking group, a set of neurons that is active at the same time.

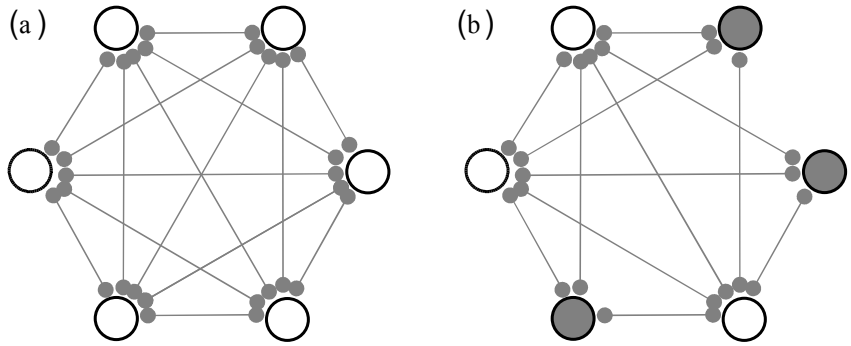

Figure 2.19: Principle of permitted sets with specific inhibitory connectivity (model a). Starting with a full inhibitory connectivity between all neurons (a), the inhibitory connections between the neurons of a permitted set are removed (b, filled neurons). These neurons form a co-spiking group. All other combinations of neurons that involve at least one inhibitory connection form a forbidden set.

Since permitted sets are not allowed to overlap, the number of permitted sets that can be implemented in a neural network is limited. Figure 2.20 shows an arrangement of ring- and cross-shape patterns in a two-dimensional grid. As a proof of concept of permitted sets, we implemented a network that exploits this structure to detect rings and crosses on the first version of our chip, see Figure 2.21 and 2.22.

In biology, neurons do not have to follow a retinotopic organization. Still, the capacity of a neural network to implement complex patterns is limited. In the example of rings and crosses, we obtain a large number of neurons that is sparsely connected. In a feedforward architecture, every pattern would be represented by only one neuron and receive a specific connectivity from the

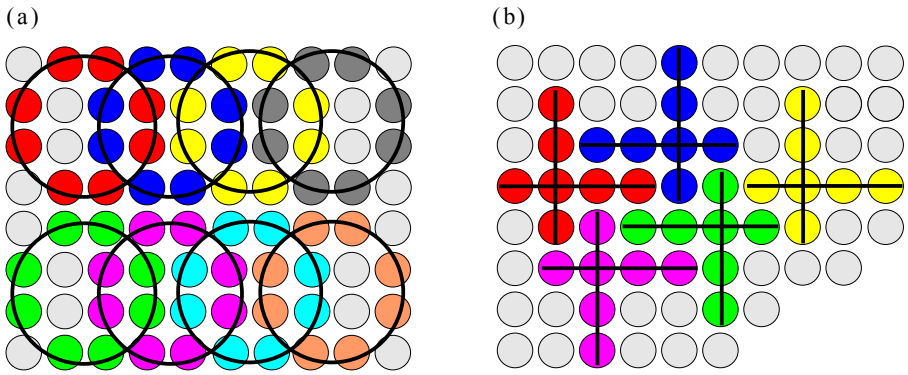

Figure 2.20: Network connectivity with permitted sets to detect (a) rings and (b) crosses. Every set of colored neurons overlaid with a black ring (cross) represents a permitted set. All inhibitory connections between the neurons of the permitted sets are removed, while each neuron inhibits every other neuron that does not belong to the same set. Since permitted sets cannot overlap, the shown setups were carefully selected to achieve a high density of sets on a two-dimensional grid. This restrictions is not necessary if neurons can be arranged in arbitrary configurations.

input layer.

It is therefore not efficient to use permitted sets in the first layer of sensory processing, but they might be useful on a higher-level of abstraction. [Hahnloser et al., 2000] proposes to model syntactic constraints with permitted sets. They suggest a network in which the neurons represent the strokes of which handwritten letters are composed. The co-activation of neurons in a permitted set would represent a letter as a combination of strokes. Combinations that do not correspond to letters are suppressed by forbidden sets. The mathematical details are similar to non-negative matrix factorization (NMF) [Lee and Seung, 1999].

As an excourse let us discuss the application of this concept of synaptic constraints to a network that plays the popular logic game 'Sudoku'. Sudoku is played on a 9x9 grid. On each grid position the digits 0-9 can be inserted, but each digit is only allowed once per line, column and 3x3 sub-grid. The player starts with some given numbers and fills in the other numbers by logical deduction. In a permitted set application, the network consists of a 9x9x10 array. Every neuron stands for a digit at a specific grid position. Inhibitory connections implement the synaptic constraints, that every digit is only allowed once per grid position, per line, column and 3x3 subgrid. Every possible solution of this connectivity forms a permitted set in the network. Obviously, the permitted sets overlap largely, so that the network has to select the permitted set that

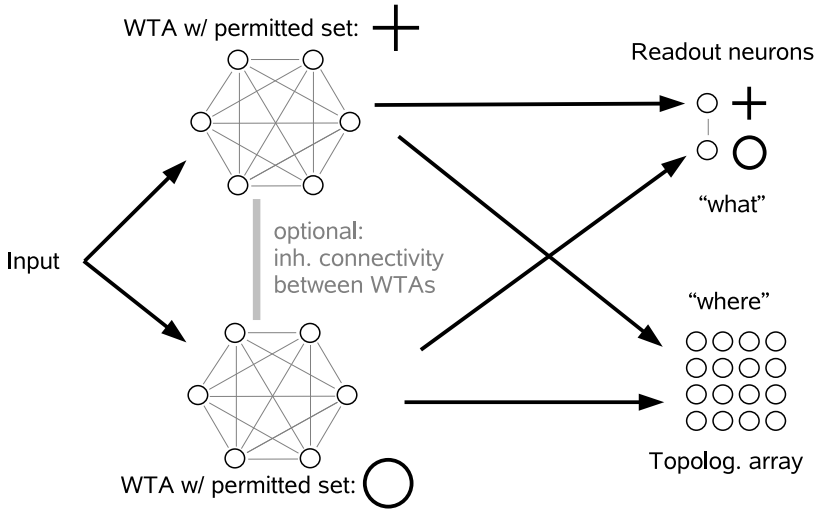

Figure 2.21: Permitted set example implementation: Network architecture. Input is connected to two winner-take-alls, one encoding for rings (top), the other for crosses (bottom), with a structure as shown in Figure 2.20. Additional connectivity between the two networks can be switched on to enforce that only one winner-take-all is active. The output consists of two readout neurons, which represent the sum of the whole activity in each winner-take-all. A second array represents the location of activity summed over the two arrays. The permitted sets detect one of the two patterns, the readout reveals the identity ("what" pathway) and its location ("where" pathway).

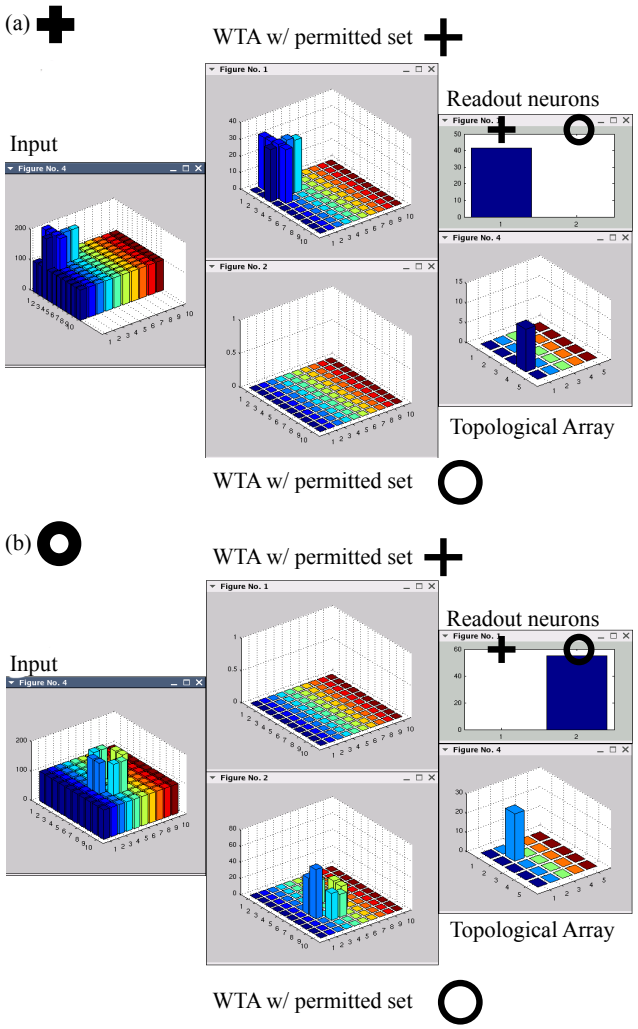

Figure 2.22: Permitted set example implementation. (b) Network Activity for the presentation of a '+'. Input consists of a cross-shape pattern overlaid with background activity. The winner-take-all with permitted sets encoding for crosses filters this pattern while it suppresses all other activity. The readout neurons indicate this with high activity at the neuron encoding for crosses. The topological array encoding for the location of the object is not functional in this implementation. (c) Network Activity for the presentation of a 'o', analogous to (b) for rings. For details of implementation, see Section 3.2.

is consistent with the given digits. We propose the following readout algorithm: background excitation drives every neuron slowly towards threshold. The neurons associated with the given digits are excited stronger so that they spike and propagate the synaptic constraints via their inhibitory connections. Neurons that do not receive inhibition will cross threshold and represent the solution of the game in their output spikes. However, in initial trials our implementation failed to settle down to a permitted solution and we could not determine if due to problems in the algorithm or the implementation.

2.5.2 Competition across Winner-take-All Networks

In the introduction we referred to the computational power of the cortical microcircuit because it processes information on several levels: within its local computational circuit, internally in the area, and inter-areas. How can competition on several levels be incorporated in our winner-take-all model?

Linear amplification means that the output of the winning neuron is proportional to its input. Our model not only has this property, but also the activity of the inhibitory neuron is equal to the sum of the outputs of the whole array. If only one neuron has active output (the winner), the firing rate of the inhibitory neuron is equal to that of the winning neuron. We can therefore use the activity of the inhibitory neuron to access the strength of the strongest input.

We combine several winner-take-all networks by adding a second inhibitory neuron to each winner-take-all (Figure 2.23). The second inhibitory neuron receives excitation from all primary inhibitory neurons, and inhibits each neuron in its associated winner-take-all array. The activity of the first inhibitory neuron is the sum of the activities of all associated array neurons. The activity of the second inhibitory neuron is the sum of the activities of all first inhibitory neurons, and therefore the sum of the activity of all array neurons in the network. The second-level winner-take-all combines all separate winner-take-all networks into one combined array.

The array neurons receive inhibition from both inhibitory neurons. In our model, all synaptic connections are hard and transmitted without significant delay through the network. If a neuron in the array spikes, its spike causes the first inhibitory neuron to spike and all array neurons receive inhibition. At the same time, the second inhibitory neuron will spike and again send inhibition to the array neurons. Since inhibition is hard, the array neurons are already discharged with the first spike and the second inhibition is not effective. For all other cases, our scheme adds an inhibitory connection between the two inhibitory neurons which belonging to a first-level winner-take-all, to compensate for the double inhibition.

In the CAVIAR project the hierarchical winner-take-all is used to determine the strongest feature. The first-level winner-take-alls determine the strongest input for each feature across the retinotopic field of view. The second-level selects the strongest feature by comparing the strengths of the winner of each

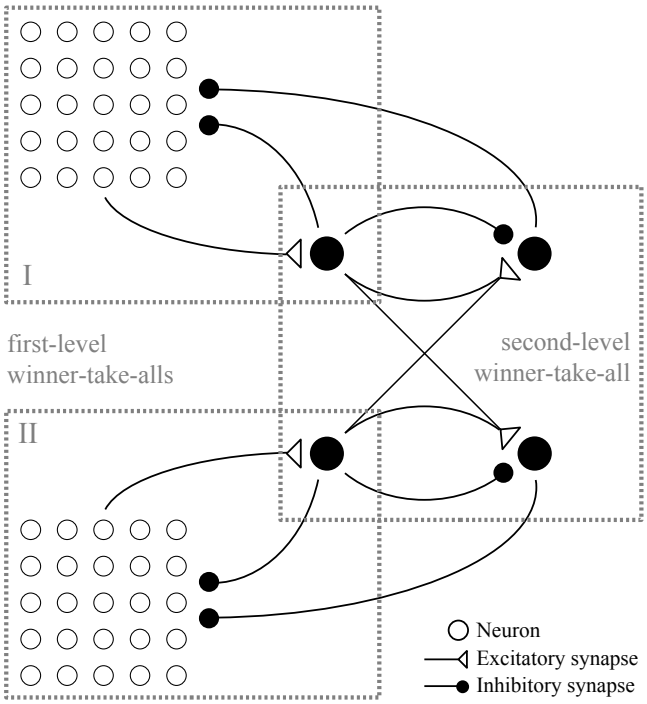

Figure 2.23: Competition across several winner-take-all networks. In our model, not only the output of the winner is proportional to the strongest input ('linear amplification'), but also the activity of the inhibitory neuron. We use this property to combine several winner-take-alls on a second level. In the first-level winner-take-all, the inhibitory neuron receives excitation from each array neurons and inhibits each neuron in return. The second-level winner-take-all adds a second inhibitory neuron to each winner-take-all. It receives excitation from all primary inhibitory neurons, and also inhibits each neuron in its associated winner-take-all. In addition, an inhibitory connection between the two inhibitory neurons belonging to a first-level winner-take-all compensates for the double inhibition.

feature map. The total winner is then the neuron with the strongest input across all features.

Cascading the winner-take-alls as presented is equivalent to one large winner-take-all that spans the total input. The separation still offers some advantages: the local winners can be determined before competition across winner-take-alls is activated, for test reasons (modular architecture). Second, the properties of the winner-take-all circuit can be different, for example their neurons have different integration parameters and different additional activity can be implemented before the winner-take-all results are merged. We will come back to this competition across winner-take-alls in the implementation Section 3.2.2.

Chapter 3

Implementation in Hardware

In this section we describe the implementation of the winner-take-all network as a hybrid analog/digital custom-made chip using Very-Large-Scale-Integration (VLSI). Neuromorphic VLSI chips implementing models of neuronal networks have a long tradition, for example see [Mead, 1989]. The goal of this chip is to implement a reliable, two-dimensional hard winner-take-all competition. Four winner-take-all networks are integrated on chip and compete against each other on a second level. In the CAVIAR project the chip is used as the 'object' chip, and the winner-take-all networks represent the feature-maps generated by the convolution chips.

We first describe the general principles of the implementation of neuronal networks in VLSI technology (Section 3.1). We introduce neuron and synapse models and how their computation is captured in analog transistor circuits. The spiking communication is captured using the address-event representation (AER), which gives the opportunity to create hybrid hardware/software frameworks using the digital communication.

In Section 3.2 we describe the details of the implementation of the winner-take-all network. In the course of the CAVIAR project, three chip versions were designed, primarily by Shih-Chii Liu. All versions share the same architecture of the winner-take-all network, but we addressed mismatch problems, improved details in the architecture and the neuron circuits of each version, and we increased the size of the population by a factor of four in the last chip revision.

We then describe the performance of the chip for the theoretical network models we developed in Chapter 2. We show the winner-take-all operation for inputs of constant currents, regular frequencies and spike trains of Poisson statistics, in comparison to the performance predicted by our theoretical model (Section 3.2.1). Competition across winner-take-alls as we discussed in the context information section 2.5.2 is shown for the example of a rotating object,

and then implemented as competition across chips for which we describe the necessary new circuits in Section 3.2.2. We then focus on two issues that have become evident in the implementation: mismatch limiting the network performance, and how spiking neuronal networks can be configured and programmed to include learning and adaptation capabilities. Mismatch (Section 3.3) limits the performance of the network. We characterize the mismatch in the input synapses and discuss different schemes of compensating the synaptic weights. We conclude with a discussion in Section 3.3.4.

Configuring and programming spike-based VLSI networks resulted in two software packages: the 'ChipDatabase' software which creates a graphical user interface for controlling the bias voltages of the on-chip circuits (described in Appendix C), and a hybrid software/hardware framework that allows to embed software agents in the spiking communication (Section 3.4). These agents analyze the spiking activity in the network, and implement a variety of learning and adaptation functions which are believed to play an important role in biological information processing.

3.1 Principles of Implementation

The cortex is probably the most complex biological structure. A recent study describes numbers of 80,000 neurons and 450 million synapses per square millimeter of cortical surface in cat visual cortex, that is every neuron receives on average 5743 synapses from other neurons [Binzegger et al., 2004]. In addition to this large connectivity, more complexity is added from the different neuronal cell types, a variety of parameters for each synaptic connection, and the detailed three-dimensional topology of each cell. How can this impressive archetype be replicated in an artificial system? The answer is, of course, by brute simplification, and by capturing two of the basic principles of neuronal systems: local computation and spiking communication.

On top of these, the dynamic properties of cortex are striking. While computation and communication happen in real-time, a variety of modulation, adaptation and learning processes take place on different temporal and spatial scales: from synaptic short-term depression and facilitation in the order of several hundreds of milliseconds to life-long changes in the connectivity patterns of brain regions due to development, training or psychological disorders. We will point out that these changes can be best explored using algorithmic descriptions, and mapped into artificial systems by embedding software agents in the spiking connectivity. For an overview of adaptive processes and the complexity of biological neurons in information processing, see [Koch and Segev, 2000].

The axon-hillock circuit, a basic neuromorphic equivalent of a neuron, was first described in [Mead, 1989], see Figure 3.1. A biological neuron is defined by its membrane, a lipid bilayer that insulates the interior of the cell from the outside medium. If ions are accumulated inside the cell, the cell has a different

potential than its surrounding, the membrane potential. In the neuromorphic implementation, the membrane is equivalent to a capacitance which accumulates charge on one plate while the other plate is connected to ground. The dielectric of the capacitance takes the role of the lipid bilayer.

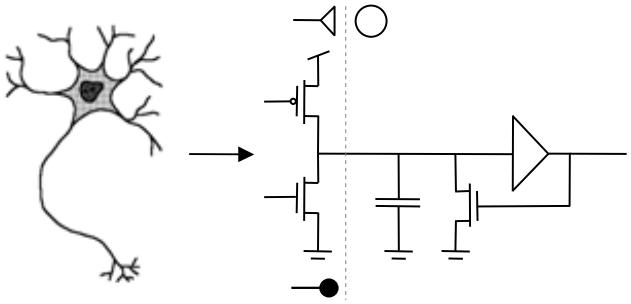

Figure 3.1: Implementation of local computation: the functionality of a biological neuron (left) is implemented in a simplified transistor circuit (right). At the dendritic tree, synapses feed input to the neuron. We use an open triangle as symbol for an excitatory synapse, a filled circle for an inhibitory synapse and an open circle for the neuron. Excitatory synapses inject depolarizing current onto the membrane capacitance. If the membrane voltage rises over a threshold, the axon hillock circuit consisting of a high-gain amplifier switches the output, and discharges the membrane capacitance through a reset transistor.

Input from other neurons is received at synapses that are arranged along its dendritic tree. Triggered by incoming pulses, channels through the membrane open and let ions flow in or out of the cell. We model ion channels by field-effect-transistors whose channels add or subtract charge from the membrane capacitance. An excitatory synapse increases the membrane voltage, an inhibitory synapse decreases the voltage.

There is a variety of ion channels that are selective for different charge carriers, and which are gated by different mechanisms, that depend on membrane voltage, time constants and chemical transmitters, resulting in very different synaptic currents. Changes in the synaptic properties are believed to be the basis of many learning and adaptation effects. Consequently, a variety of synapse models and implementations have been developed (for example [Gerstner and Kistler, 2002]). In this work we use a simple model of a synapse: the injection of a fixed current pulse onto the membrane capacitance.

Dendrites integrate the input from all synapses. In biological neurons, the structure of the dendritic tree adds to the computation by introducing delays, amplification and other non-linear effects. We consider the neuron as a point structure, that is all inputs are directly integrated on the membrane capacitance.

Once the membrane voltage rises above a certain threshold, an action potential is generated in the axon hillock. Na^+ ion channels in the membrane

open to create positive feedback, which increases the membrane potential until K^+ channels open and discharge the cell to its resting potential. The result is a sharp well-defined voltage peak, a spike. In the implementation a high-gain amplifier compares the membrane potential to a given threshold. When its output switches to high, the membrane capacitance is discharged through a reset transistor.

The action potential travels along the axon to the synapses of targeted neurons, where the process of re-transformation into a synaptic current starts again. Transmission of the action potential is a very reliable and binary process, compared to the analog computation that takes place inside the cell.

In cortex, the synaptic connections fill the three-dimensional space. Semiconductor structures are currently limited to two dimensions, in which the large connectivity can not be represented. Wiring has to be virtual with the so-called address-event representation (AER), see Figure 3.2. AER takes advantage of the fact that an action potential can be seen as a point event in time, so many events can be multiplexed over a high-speed digital bus.

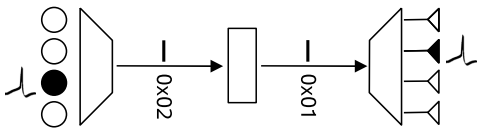

Figure 3.2: Principle of the Address-Event-Representation (AER). Whenever a neuron spikes, its address is recorded and placed as a digital event on a high-speed, multiplexed bus and transmitted off-chip. Virtual wiring translates the source address into one or several target addresses. These events are transmitted to the receiver chip, which decodes the addresses and activates the associated synapses.

On chip, the neuron circuits are repeated across a two-dimensional grid. Every neuron is assigned a row and column address. Whenever a neuron spikes, its address is placed on the multiplexed bus and transmitted off-chip, where further processing can take place. In the same way, spikes are sent to individual synapses: a decoder receives spikes on the multiplexed bus and activates the synapses belonging to the transmitted addresses. In CAVIAR, spikes can be transmitted with a bandwidth of up to 10MSpikes/s. The latency in interface hardware, for example for mapping, is less than 1.2us.

Different communication schemes have been discussed and implemented, for example [Mortara and Vittoz, 1994, Mortara et al., 1995, Boahen, 2000, Culurciello and Andreou, 2003, Boahen, 2004a,b,c]. We use an asynchronous protocol with arbitration. The arbitration ensures that a neuron can only place an event on the bus when the bus is empty. Queued neurons are processed fair and the protocol is non-greedy, that is no single neuron can take over the bus.

The actual connections are implemented virtually by mapping the source neuron addresses onto the target synapse addresses. This can be done by algorithms, or with a look-up table that lists the target synapses for every source address (Table 3.1). The table can also be used to store further properties of the synaptic connections, which are either evaluated in the mapper (for example a transmission probability), or transmitted with the spike to the target synapse (for example the synaptic weight).

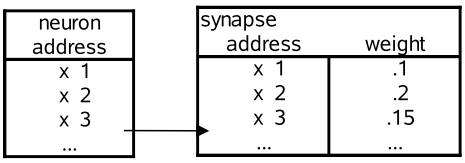

Table 3.1: Example of a connectivity table. For each neuron source address, one or several target synapses are specified. The look-up table can also contain additional parameters such as the synaptic weight or a transmission probability.

AER can be used to connect neurons and synapses on different chips, or to create recurrent connections to synapses on the same chip. Together with spiking sensors and actuators driven by spikes, complete sensory-processing systems can be assembled (Figure 3.3).

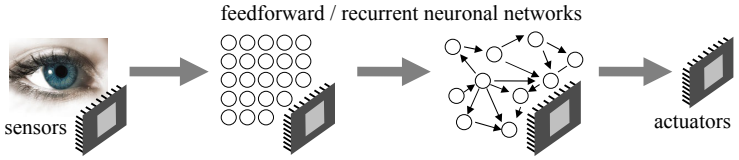

Figure 3.3: Example of an AER spike-based processing chain.

How is the higher-level functionality of learning, adaptation and modulation functions embedded into this hardware system? The functionality can be implemented in the hardware itself, for example in analog VLSI learning synapses. Since this requires substantial design effort due to the long manufacturing times, we will focus on two external solutions that are based on generic hardware.

First, the parameters of the analog computation can be controlled. Bias voltages of the circuits are used, for example, to provide a background activity to the neurons, to change the synaptic time constants or the leakage in the soma. These parameters are normally global to all neurons on one chip.

Second, the spike trains in the system can be recorded (monitoring), and artificial spike trains can be injected into the system (sequencing), that for example simulate input from higher-level areas. To interface the asynchronous

real-time spiking communication with conventional synchronous computers, a time-stamp is added to each recorded spike. For sequencing, the inter-spike interval is specified for each spike. Several solutions have been developed to interface hardware spike trains to computer systems. We use the PCIAER board [Dante and Del Giudice, 2001] or portable logic boards, controlled through the Universal Serial Bus (USB) [Rivas et al., 2005].

Based on the input from the spiking hardware, control algorithms can implement learning and modulation functions on a software base. We describe such a software system in detail in Section 3.4 [Oster et al., 2005]. The input is either activity-based, that is algorithms are based on the current estimation of the neuronal activity, or spike-based, that is computation is triggered by single events. The algorithms are implemented as independent software agents, so a variety of different algorithms can be concurrently active. Each agent acts on the spiking hardware by modifying the analog parameters, sending artificial spike trains, or modifying the connectivity table or its parameters.

	Codename	Neuron array size	AER Synapses Exc.	Inh.	Dep.
1	tsmcneuroncaviar	4x 8x8 (+ 8 gl.inh.)	2	1	1
2	tncb	4x 8x8 (incl. 8 gl.inh.)	4	2	2
3	tnc3	4x 16x16 (incl. 8 gl.inh.)	1	2	1

Table 3.2: Overview of 'Object' chip versions in the CAVIAR project.

3.2 Hardware Winner-Take-All Implementation

Our description of the network architecture is based on the principles of VLSI implementation as we introduced them in the last section. In the Introduction of Chapter 2 we discussed the general architecture of a winner-take-all network, see Figure 2.1. The network consists of a population of excitatory neurons that receives the input, and an inhibitory neuron to implement the competition. A second inhibitory neuron is used for competition across arrays as we introduced in Section 2.5.2. We will describe its implementation in Section 3.2.2.

We developed three versions of the chip, primarily designed by Shih-Chii Liu. The first two chips are fabricated in the 0.35μm process of the Taiwan Semiconductor Manufacturing Company (TSMC), the third chip in the 0.35μm process if Austria Micro Systems (AMS). On the chip the excitatory neurons are arranged in a two-dimensional grid and we will call them 'array neurons' throughout this chapter. Depending on the chip version, the size of the array is 8x8 or 16x16, see Table 3.2. Four winner-take-all arrays are integrated on one chip.

The initial plan for this chip specified two one-dimensional arrays, one for x and one for the y direction. Each of the two arrays sums the activity across rows and across columns. By this, multiple inputs can lead to the detection of spurious objects since also background activity is summed up. We therefore changed the architecture to a full two-dimensional retinotopic array.

The arrangement of the four winner-take-all networks in the three chip versions is shown in Figure 3.4. In the first two versions, the array size is 8x8. In the third version the array size is increased to 16x16. Each chip integrates four arrays, so the total resolution is 16x16 for the first two version and 32x32 for the final version. In the first version the two global inhibitory neurons of each array were placed on the side of each array, forming an additional column between the arrays, see Figure 3.4. They were not included in the AER communication, since we expected that their high output spike rate would saturate the bus. We showed in our model that the spike rates of the inhibitory neurons are equal to that of the winning neuron in the array, so they can easily be integrated into the array.

While the neurons are physically arranged in a two-dimensional grid, this does not have to reflect the actual topology of the network. Through AER

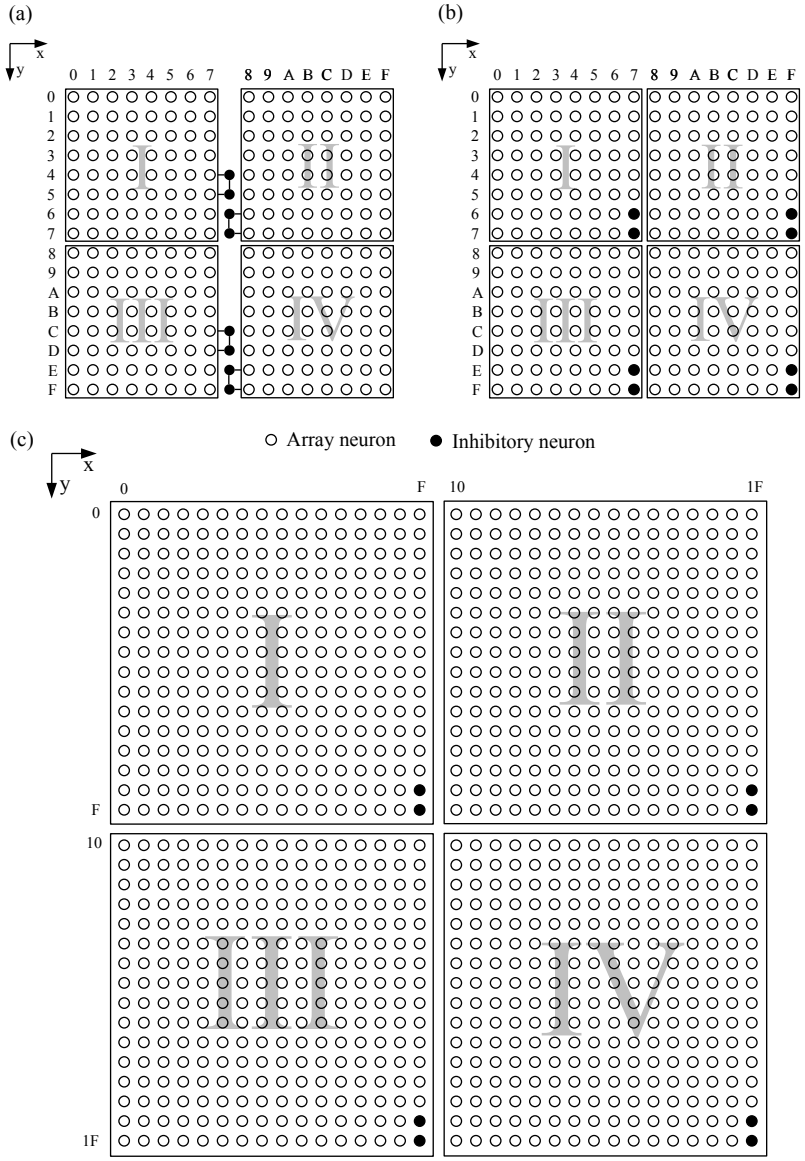

Figure 3.4: Architecture as implemented on the chip versions. Every chip version contains four winner-take-all networks (I to IV) with each two inhibitory neurons (upper one of the two: first inhibitory, lower: second inhibitory neuron). Numbers indicate the x and y AER addresses of each neuron. *(continued on page 59)*

Figure 3.4 (*continued from page 58*): (a) first chip version. The 256 excitatory neurons are tiled into four two-dimensional arrays, with the global inhibitory neurons sitting on the side of each array. They are not part of the AER communication. (b) second chip version. The inhibitory neurons are integrated into the arrays and replace two of the array neurons. (c) third chip version. The resolution of each array is increased to 16x16, resulting in a total of 1024 neurons, eight of which are inhibitory.

remapping, the input to the neurons can be connected in arbitrary topologies. Communication with the global inhibitory neurons and self-excitation is also independent of the position of the neurons in the array. Only if the local neighboring connections are used, the network topology has to follow the hard-wired connectivity.

Every array neuron receives excitatory and inhibitory AER input of different types of synapses as summarized in Table 3.2. In addition, a constant current input is used for testing. Having multiple synapses of the same type is useful when connections of different strengths are used in the network: if the synaptic weights can only be set globally for the whole chip, several synaptic weights can be chosen by addressing different synapses. In most of our experiments, a pair of excitatory and inhibitory synapses is used for the input. A second pair is used to test internal and recurrent activity. On the input synapses the weights are normally small so neurons integrate a large number of input spikes to reach threshold. The other synapses are used to explore recurrent activity such as long-range excitatory connections, or the inhibitory connectivity of permitted sets (see Section 2.5). These weights are normally higher to show a significant effect. All chip versions include a depressing synapse [Boegerhausen et al., 2003] to explore effects of dynamic synapses in the network, see Section 2.4. Since the dynamics of the synapse can be switched off it is also used as a normal excitatory synapse.

The array neurons compete with each other through inhibition. In biology, inhibition is mediated by inhibitory interneurons. In the winner-take-all network, inhibitory neurons receive excitation from the array neurons and inhibit them in return. For the theoretical analysis, the precise mechanism of the mediation of the inhibition is not important and we replaced the inhibitory neurons with a full mutual inhibitory connectivity between the array neurons. For the chip implementation, mediating the inhibition through global interneurons is efficient since these neurons sum the output of all array neurons. To implement the sum, the membrane potential node of the first inhibitory interneuron is a global line that is common to all neurons. Each array neuron has an excitatory synapse to this line to excite the interneuron, see Figure 3.5. Similarly, the spike output of the inhibitory interneuron is routed with a global line to all array neurons and forms an inhibitory synapse at each neuron ('LI4W1').

In addition to input and competition, the array neurons have hardwired local

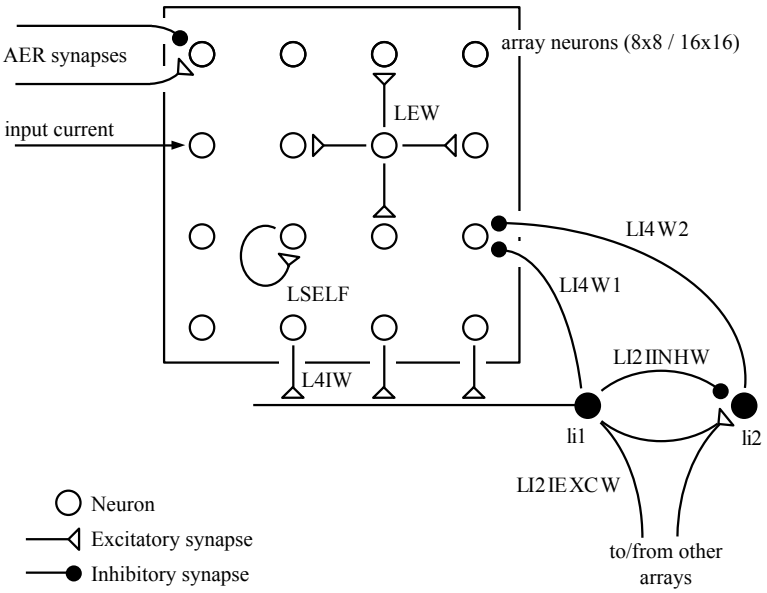

Figure 3.5: Hardwired connectivity of winner-take-all. Neurons are arranged in a two-dimensional array. Every neuron receives excitatory and inhibitory AER input, in addition to a constant input current for testing. Each array has two inhibitory neurons (li1, li2) that receive excitation from the array and feed inhibition back. The array neurons have hardwired local excitatory connections to their nearest neighbors, and a self-excitatory connection. For details, see text.

excitatory connections to their nearest neighbors ('LEW'), and a self-excitatory connection. The local-neighbor connections are intended to implement smoothing of the input, but were never used in our experiments. The self-excitatory connection ('LSELF') is a connection by which each neuron excites itself with its own output spike. This mechanism proved to be useful to stabilize the selection of the winner and we will later describe the implementation in detail.

The synaptic weights of all synaptic connections is controlled with a global bias voltage (for example 'L4IW' for the synaptic connections from array neurons to the first inhibitory neuron). If the bias is set to ground (for an excitatory synapse) or to Vdd (for an inhibitory synapse), the synaptic connection is switched off. A list of all biases and a suitable parameter range is given in appendix B.1.

The neurons implemented on chip are of the integrate-and-fire type. The neuron circuits of the first chip version have been used and described in previous work, for example [Liu et al., 2001, Boegerhausen et al., 2003]. Since the details of the circuit schematics do not affect the results discussed in this work we do not describe the details of the circuits here. Biological details to the integrate-and-fire neuron model can be found in [Gerstner and Kistler, 2002]. Note that our implementation has a constant leak rate, that is the leakage is constant, independent of the membrane potential due to leakage across the channels of the circuit transistors. The neuron and synapse circuit schematics of the second and third chip versions are modified to reduce the leakage currents; details will be published later. In the mismatch section 3.3 we will discuss the basic synapse circuit that is used on the first and the second chip version.

In the third chip version, the efficacies of the excitatory synapse and of one of the inhibitory ones are programmable using D/A converters [Wang and Liu, 2006]. We will describe their use to compensate mismatch in Section 3.3.2. These synapses are mainly used for the input, since the programmable weights can be used to calibrate the mismatch. The other inhibitory and the depressing synapse can then be used to explore recurrent connectivity.

Due to the theoretical importance of the self-excitation (see Section 2.2.1), we modified the reset mechanism of the neuron circuit. In the first version self-excitation was implemented with a synaptic connection of the output of the neuron to its input. This synapse is affected by mismatch, and interferes with the reset mechanism of the neuron soma. Starting from the second chip version, the self-excitation is included in the reset mechanism of the neuron. Instead of resetting the neuron to ground it is reset to a global bias. The global bias line is not affected by mismatch. The rising edge of the reset control signal was slowed down to decrease noise on the bias line.

In both second and third chip versions the sizes of the critical transistors have been increased to reduce mismatch. We will discuss the performance with mismatch in Section 3.3.3. We first demonstrate the winner-take-all competition as it is implemented on chip.

3.2.1 Performance

As a first test of the winner-take-all network, we stimulated the neurons with constant input currents. With these inputs and a hard winner-take-all network, the neuron with the strongest input current will always suppress the other neurons. For this experiment, we use the intrinsic mismatch of the input transistors to create a distribution of input currents. The input transistors of all neurons are driven with the same bias voltage, but due to the mismatch in these transistors, the effective input current will be different for each neuron, see Figure 3.6. The winner-take-all chip can select the strongest input current optimally, that is, it completely suppresses any other output.

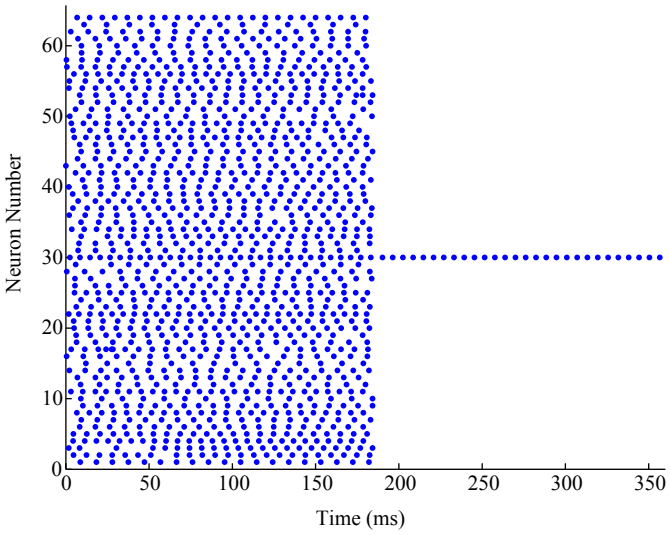

Figure 3.6: Raster plot of winner-take-all operation for stimulation with constant input currents. We use the intrinsic mismatch in the input transistors to create a distribution of input currents to the 64 neurons of the network. In the beginning, the winner-take-all connectivity is switched off. At a time of 180ms, the inhibition is activated and the neuron with the strongest input suppresses any other output. Data from second chip version.

Next, we test the response of the winner-take-all network to spike trains of regular frequencies, see Figure 3.7. As discussed in Section 2.1 the chip can detect the winner optimally, that is it selects the highest input spike rate after a predetermined number of spikes and suppresses the output from all other neurons. Again, we use the intrinsic mismatch of the chip to create a distribution of inputs. Figure 3.8 shows the time course of the winner-take-all operation.

For Poisson inputs there is a probability associated with the selection of the correct winner. As we discussed in Section 2.2, this probability depends on

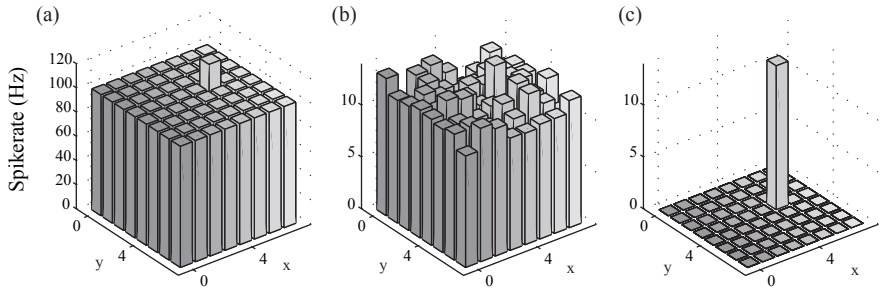

Figure 3.7: Winner-take-all operation for stimulation with spike trains of regular frequencies. The bars show the spike rates for the 8x8 neurons. (a) Input spike rate distribution. Every neuron receives a spike rate of 100Hz, except one neuron whose spike rate is increased to 120Hz. (b) Effective input, that is the output of the network without winner-take-all connectivity, reflecting the synaptic efficacies with mismatch. The neuron that receives an input spike rate of 120Hz has the strongest output. (c) Output with winner-take-all operation. The network selects the neuron with the strongest input and suppresses the output from all other neurons.

the difference in the input rates and on the number of spikes the neurons need to reach threshold. In Figure 3.9 we compare the performance of the chip to the performance predicted by the theory. The data follow the prediction of the theory, except noise in some of the data points. To reduce the effect of mismatch, we selected neurons from the chip that have synaptic weights close together, see Section 3.3.2 for a discussion of neuron sorting for mismatch reduction.

3.2.2 Competition across Chips

In Section 2.5.2 we discussed an architecture to perform competition across winner-take-all networks to create a cascade of winner-take-all networks. The inhibitory neurons of the first level compete to determine the strongest winner across all winner-take-all networks. The scheme is used in the CAVIAR project. There each winner-take-all network represents a feature-map computed by a convolution chip. On the second level of competition the strongest feature map is selected.

All versions of the winner-take-all chips contain four winner-take-all networks that cover the quadrants, see Figure 3.4. In the third version of the chip, the resolution of the sub-arrays is increased from 8x8 to 16x16.
We first illustrate the competition across winner-take-alls in the chip implementation in Figures 3.10 to 3.12.

In CAVIAR each of the winner-take-all networks corresponds to a feature-map that is computed by a convolution chip. While the numbers of sub-arrays

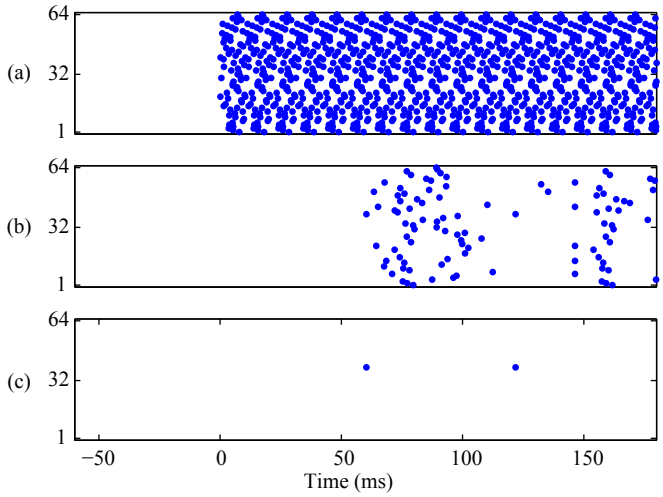

Figure 3.8: Raster plot of winner-take-all operation for stimulation with spike trains of regular frequencies. The experiment is similar to Figure 3.7, but shows the time course of the competition. (a) Input: starting from 0ms, the neurons are stimulated with spike trains of a regular frequency of 100Hz, but randomized phase to avoid synchronization effects. Neuron number 42 receives an input spike train with an increased frequency of 120Hz. (b) Output without winner-take-all connectivity: depending on the intrinsic mismatch, the neurons reach threshold with 6-8 input spikes and then spike with a regular output frequency. Neuron 42 spikes first and has the highest output frequency since it receives the strongest input of 120Hz. (c) Output with winner-take-all connectivity. Neuron 42 with the strongest input wins the computation as soon as it spikes first and suppresses all other neurons. Data from second chip version.

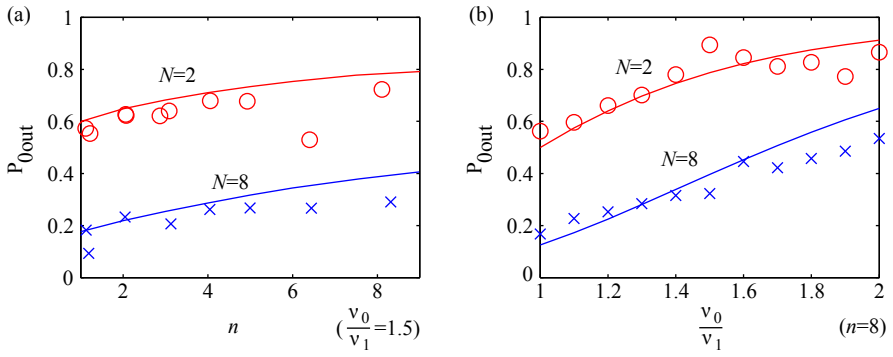

Figure 3.9: Comparison of theoretical model and results from the second chip version for stationary Poisson input. Data points show chip data, continuous lines result from the model. Circles show data for $N = 2$ neurons in the network, crosses for $N = 8$ neurons. The winning neuron receives a rate of ν_0, the non-winning a rate of ν_1. In the case of 8 neurons, all non-winning neurons receive the same rate ν_1. (a) Probability of correct output versus the average number of spikes the neurons need to reach threshold n. We give an average number n because of the mismatch in the synaptic efficacies. The difference in the input rate of the winning neuron ν_0 versus the non-winning one(s) is 50%. (b) Probability of correct output versus the ratio of the winner input rate to the non-winning input. The neurons need 8 spikes to reach threshold. In both cases, the data follow the prediction of the theory with single outliers due to noise, for example in (a) for $N = 2, n \approx 6.5$.

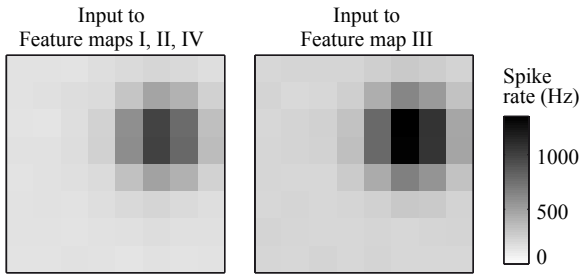

Figure 3.10: Competition across four feature maps. Input to the neurons in each feature map consists of spike trains of regular frequency and represents a two-dimensional Gaussian distribution that rotates around the center of the array with a frequency of 0.25Hz. The distribution is scaled so that the highest input rate equals 1000Hz (this high rate is necessary because the network has to determine the winner using an estimate of the instantaneous input rates on the moving stimulus). In addition, every neuron receives a background firing of 200Hz. The input to feature map III is increased by 35% compared to the input spike rates to feature maps I, II and IV.

on the object chip is fixed to four, the number of features at the convolution stage is variable, given by the number of convolution chips. For example, in the second assembly of the CAVIAR chain, only two convolution chips were used which left half of the neurons of the object chip unused. We therefore explored a scheme in which the competition across arrays can be performed as competition across chips, see Figure 3.13. Every chip corresponds to one feature map. This architecture is more flexible, since it accounts for a variable number of convolution chips, and increases the resolution of each winner-take-all by four since now the complete chip is used for a winner-take-all network and not only a quadrant. The third version of the object chip then provides a resolution of 32x32 per feature map, matching that of the convolution chip.

Across-chip competition is implemented with a wired-OR circuit, that is a wire common to all winner-take-all chips. On each chip, the four sub-arrays are combined to form one large array by combining the output of the first inhibitory neurons with an OR-circuit in the signal 'CompAcrossOut', see Figure 3.13. Several chips compete by combining their 'CompAcrossOut' signals with a wired-OR. A resistor is placed on the output to pull the signal to Vdd when no chip drives the 'CompAcrossOut' line. The circuit for the OR gate with open drain is shown in Figure 3.14.

The common 'CompAcrossOut' signal is the input to the second inhibitory neurons of all chips ('CompAcrossIn'). Since spikes from several chips in the wired OR can collide, we added a monostable circuit, see Figure 3.15. The design of the hard winner-take-all network prevents these collisions: as soon as

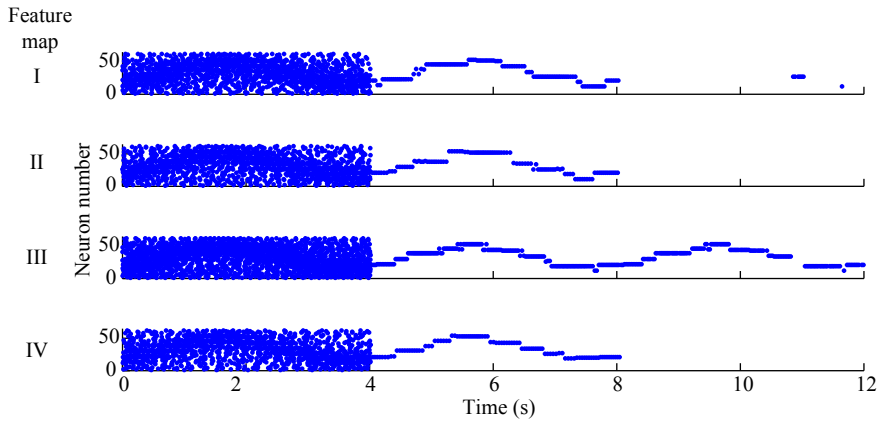

Figure 3.11: Competition across four feature maps. Rasterplot of the output for the four feature maps. Each dot represents one spike. From $t=0$ to 4s, the winner-take-all connectivity is switched off and the neurons respond to the input with the rotating Gaussian peak and a background firing. At time $t=4s$, winner-take-all competition takes place within each feature map. All neurons except the winners are suppressed. The position of the winning neurons follow the center of the Gaussian peak, which rotates once in 4s. At time $t=8s$, the competition between feature maps is switched on by activating the connections to the second inhibitory neurons. Only feature map III that receives the strongest input is active and suppresses the neurons in the other three feature maps. The output of the inhibitory neurons is not shown. Data from second chip version.

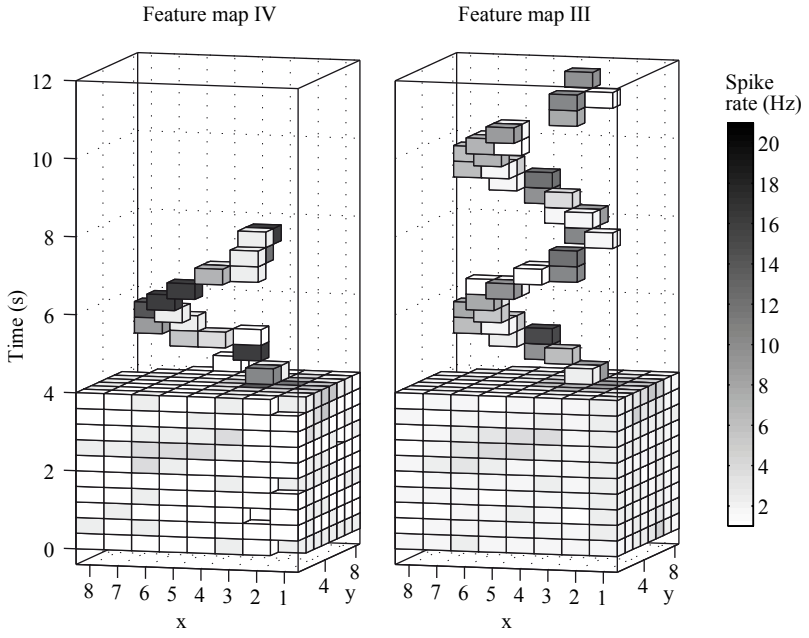

Figure 3.12: Competition across four feature maps. Time-binned output for two feature maps. The three-dimensional plot shows the activity of the neurons (x, y) over time (z-axis). The bin time is 40ms. A cube is shown if the pixel spiked at least once in the given bin. Connectivity and time course as explained in Figure 3.11. Only feature maps III and IV are shown. Data from second chip version.

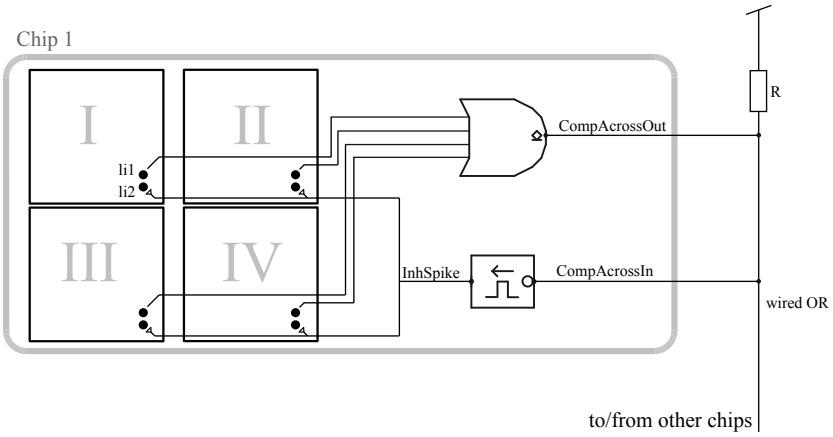

Figure 3.13: Competition across chips. In the on-chip competition, each of the four sub-arrays I-IV on the chip represent a feature map (see Figure 2.23). In the competition across chips, the four sub-arrays are combined and the whole chip represents one feature map. To combine the sub-arrays, the output of the four first inhibitory neurons li1 is or'ed together to the signal 'CompAcrossOut'. To implement competition across chips, this signal combines the output of the first inhibitory neurons on other chips in a wired OR. The 'CompAcrossOut' signal is open-drain and the wired-OR contains an external resistor R that brings the node to Vdd if no chip is driving it to ground. The input 'CompAcrossIn' contains an impulse former and stimulates the second inhibitory neurons that inhibit the array neurons of all sub-arrays. Seen from the outside, the chip contains one winner-take-all including all neurons instead the four sub-arrays. This scheme provides more flexibility than the competition on-chip since the number of object chips can match that of the feature maps instead of having a fixed number of sub-arrays on one chip. In addition, we get more resolution because the number of neurons per feature map is increased by a factor of four.

the array neurons receive inhibition, they are reset so the inhibitory neurons are not driven and the external line is not driven until one of the array neurons will cross threshold again. The wired-OR circuit is therefore an easy and simple solution to implement competition across winner-take-alls across chips.

We designed an interface board to contain up to four winner-take-all chips that are connected with the wired-OR circuit, see Figure B.3. This board contains some shorts due to manufacturing problems. When inserting more than one chip, the board goes into power oscillations. We were not able to discover if the source of this error is due to the manufacturing problems or problems in the digital interface logic.

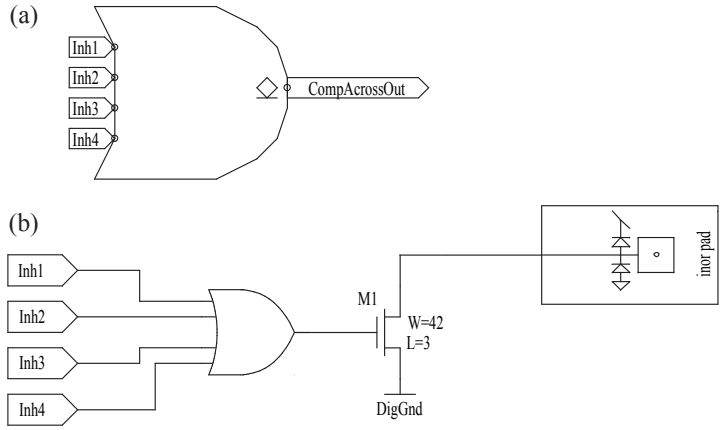

Figure 3.14: Competition across chips: output circuit. (a) Circuit symbol, combining the output of the four inhibitory neurons ('Inh1' to 'Inh4') in an OR-gate with open-drain ('CompAcrossOut'). (b) Implementation. The spiking output of the inhibitory neurons is combined with a standard OR gate. The output of this gate drives the open-drain transistor M1. Width and length of the transistors are chosen to sink a large external current. The transistor is connected to an input pad, although logically the 'CompAcrossOut' signal is an output of the chip.

With this section we finish our description of the implementation of the winner-take-all chip. The remaining sections focus on specific topics of implementation: a characterization of mismatch and mismatch compensation (Section 3.3), and a hardware/software framework to embed learning in spiking systems (Section 3.4). Further details of the different versions of the winner-take-all chips such as the bias settings and the PCBs can be found in Appendix B.

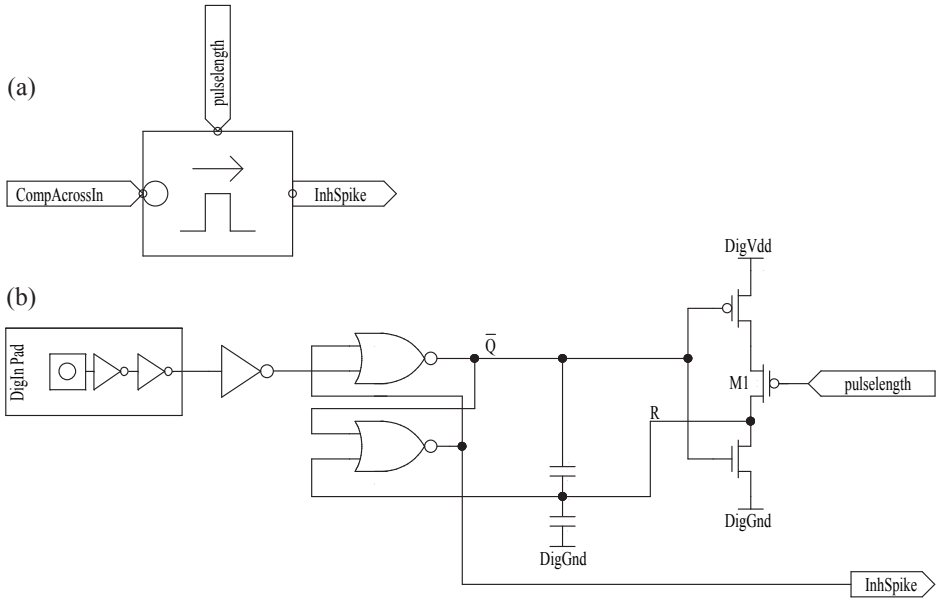

Figure 3.15: Competition across chip: input circuit. (a) Circuit symbol. Pulses on the input of the external line ('CompAcrossIn') are formed with a monostable to have a defined pulsewidth that can be adjusted by a bias ('pulselength'). The output of the monostable drives the inhibitory synapses of all array neurons.

3.3 Mismatch and Synaptic Weights

In neuromorphic chips, neurons and synapses are integrated in large arrays on the chip. The large area covered by the array makes the individual circuits sensitive to variations in the parameters of the manufacturing process. These variations cause variations in operating parameters of the circuits, especially in the threshold voltage of the transistors. Analog transistor circuits operating in the subthreshold regime are especially sensitive to this mismatch. The result is a large fixed-pattern noise in the synaptic properties which limits the functionality of the chips.

To understand the basics of mismatch in analog VLSI circuits we will first characterize the mismatch on the level of a single transistor by looking at the effect of a single transistor that injects a constant input current into the neuron, and measuring the output rate of the neuron. Compared to other characterizations that use special test structures to quantify mismatch in transistors, for example [Serrano-Gotarredona and Linares-Barranco, 1999], our measurements use the integrate-and-fire neurons themselves as the output device. While the measurement accuracy is lower, this is actually an advantage since the mismatch is characterized with respect to the functionally relevant output of the whole circuit. We compare data from the first two versions of the winner-take-all chip. We then extend our analysis to spiking input, since this type of synapse is used in the network implementation.

Calibrating mismatch in the synaptic weights is equivalent to setting all synaptic weights to the same value. We will discuss five different strategies to set the synaptic weights in the context of mismatch compensation.

3.3.1 Mismatch Characterization

Constant Input Current

The circuit in which we perform our measurements is shown in Figure 3.16.

The synapse transistor M1 inserts a constant input current I_{M1} that charges the membrane capacitance C_{mem} of the neuron. When the membrane potential crosses the threshold V_{th} after a time Δt, a spike is fired and the neuron is reset to V_{reset}. We measure the output spike rate $f = 1/\Delta t$ for each neuron and calculate the input current:

$$C_{mem} = \frac{I \cdot \Delta t}{V_{th} - V_{reset}} \Leftrightarrow I = C_{mem}(V_{th} - V_{reset})f \tag{3.1}$$

We assume C_{mem} to be constant, since the mismatch of the poly1-poly2 and moscap capacitances is reported to be small in this technology. In [Oster and Liu, 2004] we showed that the variation of V_{th} and V_{reset} does not contribute significantly to the variation of the firing rates, and that the effect of the refractory period can be neglected. We therefore assume V_{th} and V_{reset} to be the same for all neurons, see Table 3.3.

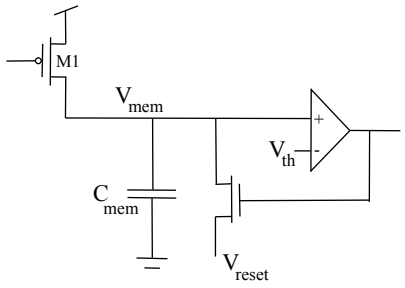

Figure 3.16: Simplified version of the neuron circuit used to measure the constant input current. Transistor M1 injects a current onto the membrane capacitance C_{mem}. If the membrane voltage V_{mem} rises over a threshold V_{th}, the output is switched high (in the implementation a diff-pair is used), and discharges the membrane capacitance through a reset transistor to the reset voltage V_{reset}.

The second version of the winner-take-all chip has two input transistors M1 and M2 with different geometries, see Table 3.3.

		Chip I	Chip II	
			M1	M2
C_{mem}	[fF]	62.89	270.8	
V_{th}	[V]	1.70	1.70	
V_{reset}	[V]	0	0	
W×L	[μm]	1.2×1.2	2×2	1.2×1.2
U_T	[mV]	25.8		

Table 3.3: Constants and specifications.

The effective current I_i of neuron i charging the capacitance is the difference between the input current I_{M1} and the leak current I_{leak} of the soma:

$$I_i = I_{\mathrm{M1}} - I_{leak} \tag{3.2}$$

Transistor M1 is operated in the subthreshold regime, so we get:

$$I_i = I_0 \cdot \exp\left(\frac{-\kappa V_{gs}}{U_T}\right) - I_{leak} \tag{3.3}$$

V_{gs} is relative to V_{dd} and is the gate to source voltage of M1.

Both I_{M1} and I_{leak} differ for each neuron i because of mismatch. In Figure 3.17 the current I_i is shown versus V_{gs} of M1 for the neurons with the largest and the smallest firing rate. The subthreshold slope factor κ is equal for both transistors, but the prescaling factor shows large variations. For low currents, the leakage current is the dominant factor.

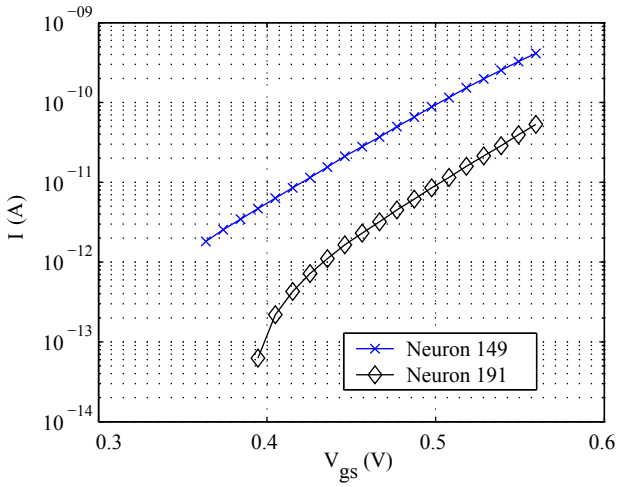

Figure 3.17: I vs. V_{gs} for the two extreme input current transistors.

To determine the variation of the input current through transistor M1 over all neurons, that is I_0 in Equation 3.3, we choose V_{gs} so that all transistors are still operated in the subthreshold regime and are not affected by I_{leak}. The resulting variations measured for both versions of the chip are summarized in Table 3.4.

		Chip I	Chip II	
		M1	M1	M2
κ	-	0.798	0.70	0.70
$\sigma(\Delta x/\langle x\rangle)$				
$I_{\mathrm{M1/2}}$	%	62.3	19.4	38.1
$\sigma(V_{T0})$	[mV]	18.2	6.1	11.5
A_{Vth}	[μm·mV]	21.8	12.2	13.8

Table 3.4: Measurements and reference values.

The distribution of the prescaling factors I_0 does not follow a Gaussian distribution (Figure 3.18). The prescaling factor is given by:

$$I_0 = I_s \exp\left(\frac{\kappa V_{T0}}{U_T}\right) \tag{3.4}$$

We fitted the I_0 distribution by assuming a Gaussian distribution of the threshold voltage. We calculated V_{T0} from Equation 3.3 with $I_{leak} = 0$ and $I = I_{th}$

defined by [Eisele, 1998]:

$$I = I_{th} \equiv \frac{W}{L} \cdot 0.1\mu A \tag{3.5}$$

The parameters $\mu(V_{T0})$, $\sigma(V_{T0})$ and I_s were then estimated from the I_0 distribution using least-square optimization. Table 3.5 lists the parameters of the fit.

	$\mu(V_{T0})$ [V]	$\sigma(V_{T0})$ [mV]	I_s [nA]
fit	0.603	11.5	3.7

Table 3.5: Fitted parameters of I_0 distribution, from a least-square fit assuming Gaussian distribution of V_{T0}.

I_s is then calculated from Equation 3.4 using these mean values for I_0 and V_{T0}. Figure 3.18 shows that the measured distribution of I_0 is modeled well from the assumption of a Gaussian distribution for V_{T0}.

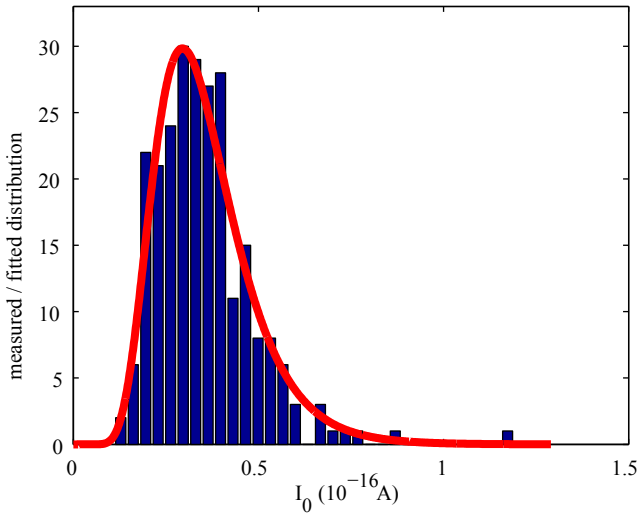

Figure 3.18: I_0 distribution measured (histogram) and fitted (continuous line).

We also determined the mismatch matching parameter A_{Vth}, which differs largely for the two chip versions and is in all cases larger than reported for a generic 0.35μm CMOS technology [Eisele, 1998].

$$A_{Vth} = \sigma_{Vth} \cdot \sqrt{W_{eff} L_{eff}} = 36.5mV \cdot \mu m \tag{3.6}$$

Both the fits of the I_0 distribution and the parameter A_{Vth} suggest that there are additional mismatch sources other than the threshold voltage variation given

by Equation 3.4. For example, the model proposed by [Serrano-Gotarredona and Linares-Barranco, 2000] uses five different parameters to model transistor mismatch.

Parasitic differential capacitances do not contribute to the experiment here, since they only have an effect when the input is switched on or off. We stimulated the neurons with a constant input current which should not induce dynamic effects.

Figure 3.19 shows the variation for the two transistors across the population of neurons. For high firing rates, the mismatch is determined by the prescaling factor I_0 and can be roughly assumed as independent from the mean firing rate. For low firing rates, the leakage current is the dominant factor and the total variation is determined by the higher mismatch of the leakage currents.

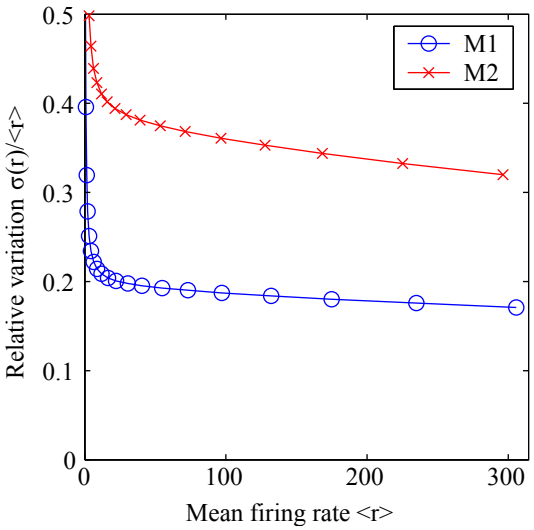

Figure 3.19: Relative variation of the output firing rate. Data is obtained for the two transistors that charge the membrane capacitance with a constant input current. We vary the input current to obtain different mean firing rates. The relative variation is the standard deviation of the firing rate distribution divided by its mean. Data from the second chip version.

Spiking Synapse

Next we measure the mismatch in the real synapse that is driven by input spikes. Figure 3.20 show the synaptic circuit. An incoming spike, consisting of a short pulse of about 1μs, switches transistor M2 on and copies the current I_{syn} set by transistor M1 with bias V_w to the membrane capacitance.

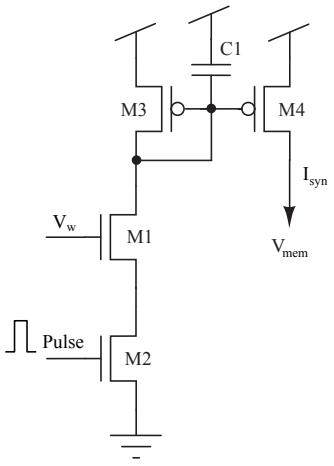

Figure 3.20: Spiking synapse circuit.

To characterize the synapses we stimulate them with a spike train of constant frequency r_{in} and measure the output frequencies r_{out} of the neurons. We define the dimensionless synaptic efficacy w which describes the jump in the membrane potential relative to $V_{th} - V_{reset}$. Note that we do not consider conductance-based synapses or the time course of the synaptic current here. A synaptic input causes a fixed, instantaneous jump in the membrane potential. For example with a synaptic efficacy of w=0.2 the neuron will reach threshold after 5 input spikes.

The neuron will only cross threshold and make an output spike at the time it receives an input spike. To avoid that this synchronization has effects on the accuracy of the measurement, only small values of w can be measured (Figure 3.21). To avoid influence of the soma leakage on the measurement, we use a high stimulation frequency, so that the membrane time constant can be avoided.

$$w = \frac{r_{out}}{r_{in}} \quad \text{for} \quad r_{out} < 0.1 \cdot r_{in}, \quad r_{in} \gg \frac{1}{\tau_{leak}} \tag{3.7}$$

We use r_{in}=2.5kHz and characterize the synapses of the 248 excitatory neurons on the chip in blocks of 62 to avoid saturation of the communication bus.

Figure 3.22 shows the relative variation of the synaptic efficacies. The properties are similar as the ones discussed in Fig 3.19, but the leakage range is less pronounced.

3.3.2 Compensation Schemes

How can the mismatch be compensated? In this section we discuss and compare several methods. In general, mismatch can be reduced by appropriate sizing of the critical transistors. The models described in [Serrano-Gotarredona and

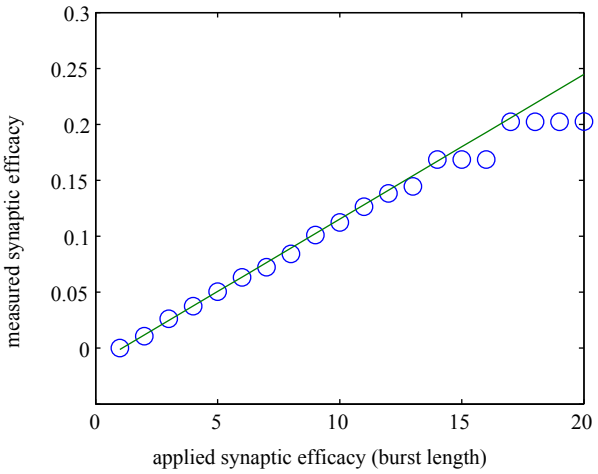

Figure 3.21: Effect of synchronization by stimulation with input spikes. The neuron will only cross threshold when it receives an input spike. For high synaptic efficacies the output of the neuron is synchronized to the input. for example the neuron can only reach threshold with 4 or 5 spikes, but not with 4.5 input spikes. Shown is the measured synaptic efficacy as defined in Equation 3.7. The applied synaptic efficacy is increased using the burst length adaptation described in section 3.3.2.

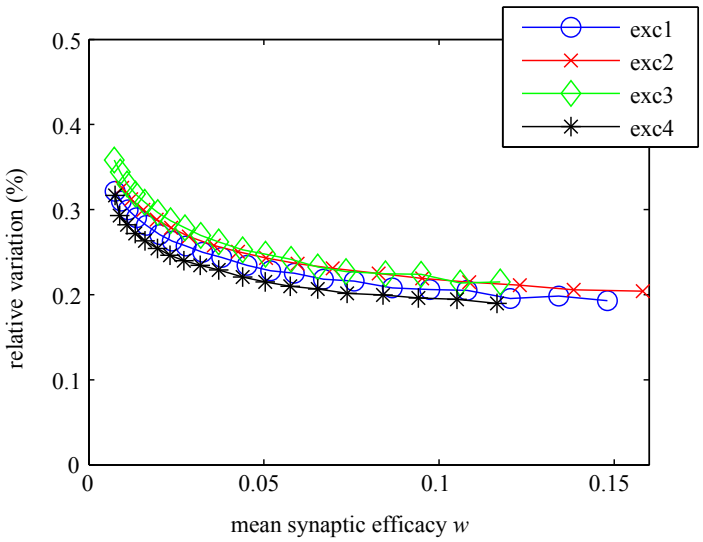

Figure 3.22: Relative variation of the synaptic efficacies for different mean firing rates. Data from 4 independent excitatory synapses of the 2nd chip version is shown ('exc1'-'exc4'). The transistor M1 of synapse 'exc2' has double the width than the other synapses. Since the mismatch is not reduced for this synapse, the most probable mismatch source are the transistors of the current mirror, see Figure 3.20.

Linares-Barranco, 2000] and [Drennan and McAndrew, 2003] provide enough detail to predict the amount of mismatch on-chip after fabrication from known transistor widths and lengths.

A different scheme is to alter the synaptic efficacies at run-time. Changing the synaptic efficacies at run-time is also required for any implementation of networks with graded connections, and for systems that use adaptation or learning (we will discuss network models that exploit the intrinsic mismatch at the end of this section).

Synaptic weights can be set on-chip or off-chip. On-chip synaptic weight setting is normally used when learning algorithms are implemented on-chip. The synapses use local learning rules, for example spike-timing dependent plasticity (STDP). The synaptic efficacy is either stored using short-term analog memory, optionally with multistable states [Bofill-i-Petit and Murray, 2003, Riis and Hafliger, 2004, Arthur and Boahen, 2004, Mitra et al., 2006], or using floating gates [Diorio et al., 1996]. Such synapses incorporate biological features quite well, for example they do not separate the memory from the processing unit. Although these synapses are biologically inspired, they have a major drawback in a chip implementation. In the AER communication, synapses can be multiplexed, that is multiple connections are implemented by sending spikes from different sources to the same target synapse. If the synapses stores its parameters internally, each synapse has to be implemented separately resulting in large area consumption. [Mitra et al., 2006] reports a chip with 256 synapses per each of the 32 neurons, in which every synapse has its own analog weight memory.

In an AER system, the synaptic connectivity of the network is implemented by a look-up table that routes spikes from neurons to synapses and is stored externally. It is natural to use this external memory to store the parameters of the synaptic connections, since digital memory is cheap and available in large sizes. The interface logic either processes the synaptic parameters during routing (for example for stochastic spike transmission) [Mallik et al., 2005], or it transmits the parameters to the chip for further processing. A special case is [Serrano-Gotarredona et al., 2006] in which the synaptic weight is programmed every time a convolution kernel is applied. The synaptic weights are loaded from on-chip RAM memory that is integrated on-chip.

Storing the synaptic weight externally is useful if learning and adaptation algorithms are implemented in software as we will describe in Section 3.4. In that case, the synaptic parameters are updated directly by the algorithm.

In this analysis we focus on mismatch in the synaptic efficacies, although all parameters of a synapse can be affected by mismatch, for example the time constants. The synaptic efficacy is the most important parameter in our network model. Compensating the mismatch in other parameters of a synapse would require additional schemes to the ones described here.

We will discuss five different schemes to set the synaptic weight through external logic. All methods can be used to compensate the mismatch intrinsic to the chip implementation. In a neural network, the weights are initialized

to the distribution given by the network model, and then modified through learning. For mismatch compensation, the synaptic weights are set to be as equal as possible across the whole population. We will use this special weight distribution as an example case to discuss the weight setting algorithms.

Neuron Sorting

A first approach is to use only the neurons whose efficacies are close together and to discard the neurons with outlying efficacies. This is inspired by a biological principle that a variety of samples is grown and the non-suitable ones are discarded, for example for the growth of axons, for genetic selection etc. In the case of an aVLSI chip, the virtual AER wiring is altered to exclude the neurons that are sorted out. How effective is such a strategy to reduce mismatch?

Starting with the skewed Gaussian distribution of firing rates measured from the first version of our chip (Figure 3.23a), we sorted the neurons by the difference of their firing rate to the mean firing rate of the population. We discarded the neurons at the end of this list, that is the neurons with outlier firing rates. The coefficient of variation (CV) of the firing rates of the remaining population is shown in Figure 3.23b. In the beginning the method is more efficient than a linear decrease, that is by discarding only 5% of the neurons in the population, the CV drops from initially 33% to 28%. Then the coefficient of variation decreases slower. We compared our result to that of a non-skewed Gaussian distribution. The method is slightly more efficient on a skewed Gaussian distribution, since there are more far outliers than in the Gaussian distribution.

Although the variation decreases in a superlinear way, neuron sorting is not as efficient as we expected in reducing mismatch. Still, by discarding a small percentage of neurons in the population the variation can be significantly reduced. Neuron sorting is only possible if the wiring within the network is established with the AER. If the network is hard-wired, selecting only a subset of neurons is not possible. We use neuron sorting whenever we implement networks that do not require the full number of neurons on the chip.

Burst-Length Adaptation

In burst-length adaptation a burst of spikes is sent to the synapse instead of a single spike. The length of these bursts is adapted to set a specific synaptic weight. The bursts are generated externally with the mapper interface. The method has the advantage that it makes use of existing off-chip hardware, does not require additional on-chip circuits, and is easy to implement.

We present results from the first chip version, showing that the method reduces the variation in the synaptic efficacies from about 100% to 10%.

The burst is generated by repeating the same target address in the routing table of the external spike interface module. In the PCIAER board the address of the synapse is repeated in the list of target addresses. The USBAER mapper

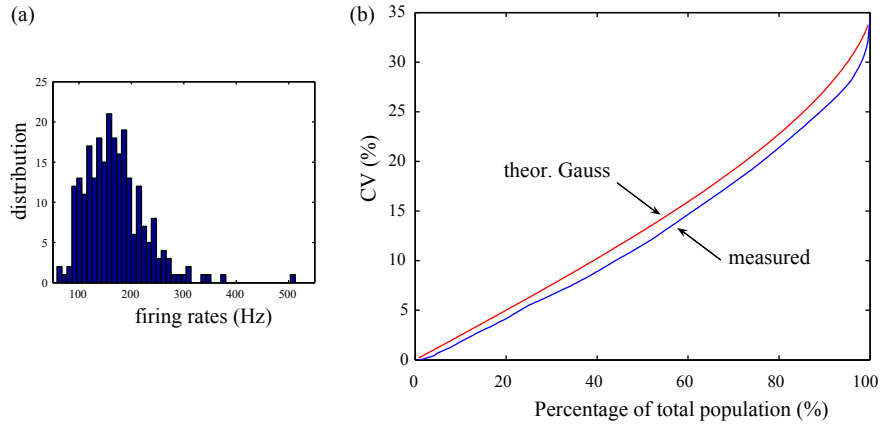

Figure 3.23: Mismatch compensation with neuron sorting. Starting from the firing rate distribution of all neurons (a), only a subset of the neurons which have similar synaptic efficacies are kept (b). The coefficient of variation decreases as part of the neuron population is used. Data from second chip version, see inlay for distribution of firing rates for stimulation with constant input currents.

has a special entry in the mapping table that specifies how often the target address should be repeated.

We first checked that multiple spikes sum linearly on the membrane potential (Figure 3.24). The time constant of the synapse has to be shorter than the cycle time of the handshake on the bus (note that this requirement excludes synaptic models that rely on specific dynamics of the synaptic current).

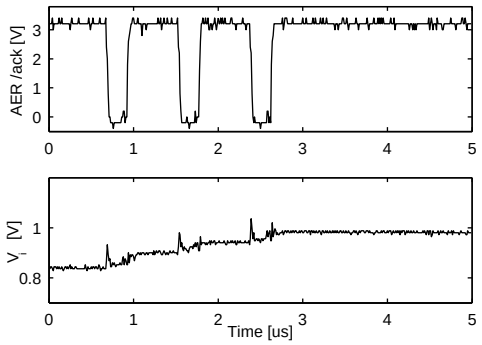

Figure 3.24: Burst of spikes sum up linearly on the membrane potential, shown for 3 spikes. (a) one of the AER communication signals indicating that the chip has received a event from the interface module (active low acknowledge signal from the chip); (b) membrane potential V_k of the neuron. With every incoming spike the membrane voltage is increased. The charging of the membrane takes less time than a communication cycle, so subsequent spikes sum up independently.

We vary the number of spikes in the burst m_i for each neuron until the neuron spikes with the desired output frequency. The output frequency can only be an integer divider of the input frequency, because the neuron needs an integral number of spikes to reach threshold. We used a simple algorithm to adjust m_i: all burst lengths are initialized with the same lengths, and the firing rates are measured. If the output frequency of a neuron is below the average frequency, m_i is incremented by one, otherwise it is decremented. The process is iterated until the coefficient of variation saturates.

We tested this method on the first version of our chip which exhibited the largest amount of mismatch. Uncalibrated, the vector of output spike rates $\mathbf{r}$ has a mean of $\mu_{\mathbf{r}} = 11.28\text{Hz}$. On average a neuron needs $\langle n_i \rangle = 8.86$ input spikes to reach threshold. The standard variation of the output rates is $\sigma_{\mathbf{r}} = 11.69\text{Hz}$ and the coefficient of variation $\text{CV} = \sigma_{\mathbf{r}}/\mu_{\mathbf{r}} = 103.7\%$.

Figure 3.25 shows the resulting output firing rates after mismatch compensation. The mean frequency $\mu_{\mathbf{r}} = 11.60\text{Hz}$ is equal for the calibrated and the uncalibrated case, but the standard deviation is now only $\sigma_{\mathbf{r}} = 1.006\text{Hz}$, resulting in a coefficient of variation of 8.6%.

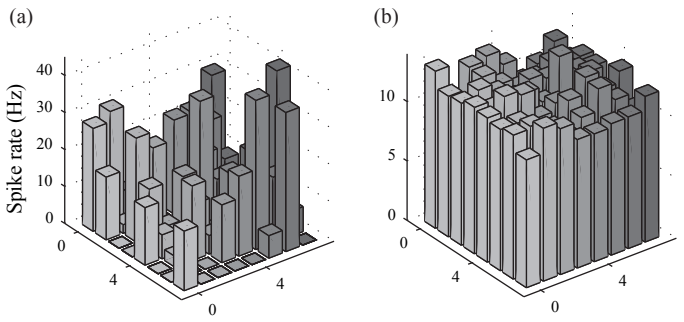

Figure 3.25: Output firing rates of the 8x8 neuron array, (a) without and (b) with mismatch compensation using burst-length adaptation. Each neuron is stimulated with a constant input frequency of 100Hz. X/Y axis: neuron address; bar height: spike rate.

The first chip exhibits also mismatch in the reset mechanism of the neurons. Because of this mismatch, some neurons are not completely discharged while others show a finite refractory period (Figure 3.26). To correct for this mismatch with the burst-length adaptation method, we used the internal reset mechanism of the neurons to only discharge the membrane potential by a small voltage determined by the hysteresis capacitance. We then sent a burst of p_i inhibitory spikes to a neuron, triggered by the output spike of the neuron. The number of spikes p_i is adjusted so that all neurons are reset to the same voltage (Figure 3.27).

On the first chip there is mismatch in the reset mechanism of the neurons since the same circuit is used to control the discharge of the membrane capacitance and to set the refractory period. The circuit uses a source biase to set the time constant, resulting in large variations. On the second chip version we changed the circuit to have independent control of membrane reset and the refractory period, so the neurons were not subject to this type of mismatch anymore.

Despite its effectivity in reducing the mismatch, burst length adaptation has the disadvantage that it requires a lot of bandwidth on the bus. Instead of sending a single spike to the target synapse, multiple spikes have to be transmitted. The average number of spikes in the bursts determines the bandwidth taken up on the AER bus.

The resulting variation depends on the average number of spikes in the burst. The efficiency of the algorithm decreases as the average number of spikes in a burst is increased, that is the achieved mismatch reduction starts to saturate, see Figure 3.28. A second effect is that the mismatch reduction is most efficient for a large variation in the uncalibrated case.

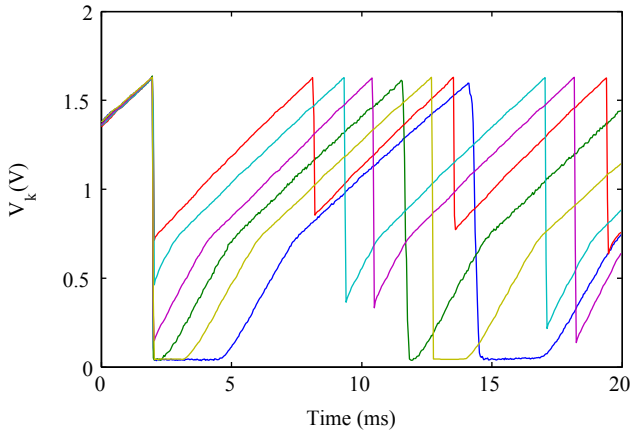

Figure 3.26: Mismatch in the reset voltage of the neurons, shown for some example traces. Some neurons are not completely discharged while others show a refractory period. Traces from first chip version.

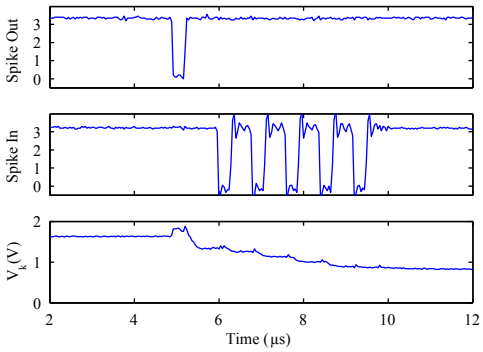

Figure 3.27: Compensation procedure of the reset voltage mismatch. An output spike of the neuron (a) is routed back by the external mapper as a burst of 5 spikes to the inhibitory synapse of the neuron (b), discharging the membrane potential (c) to a defined reset voltage.

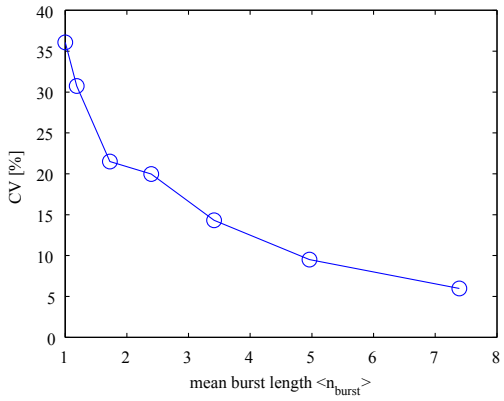

Figure 3.28: Mismatch reduction using burst-length adaption. Shown is the coefficient of variation versus the average number of spikes in a burst (data from second chip version). For each data point, the burst lengths to each of the 256 neuron were adjusted to reduce the coefficient of variation of the population. The mismatch reduction is most efficient for high rates of variation in the uncalibrated state. For larger number of spikes in a burst the mismatch reduction starts to saturate.

To quantify this behavior, we developed an optimization procedure to estimate the minimum variation depending on the average number of spikes in a burst in the calibrated case. The average burst length determines the bandwidth that is taken up on the communication bus for mismatch compensation. The algorithm is detailed in Figure 3.29.

Figure 3.30 shows the minimal variation as optimized by the algorithm, depending on the average burst length in the calibrated case. The real chip data (Figure 3.28) shows more variation than the theoretical optimization, since also other factors contribute to the mismatch in the neuron firing rates.

D/A Converter

Digital-to-analog (D/A) converters are used to directly program the synaptic weight at the time the synapse is activated. Similar to burst-length adaptation, the synaptic weight is stored together with the target address of the synaptic connection in the mapping table. When the connection is activated, that is a source address that encodes for one or several connections is received, the mapper sends two spikes to the target synapse. The first is a special address that allows programming the D/A converter, the second is the address of the synapse itself. We had to use two consecutive spikes since the number of bits in the address was not large enough to hold both D/A value and synapse address.

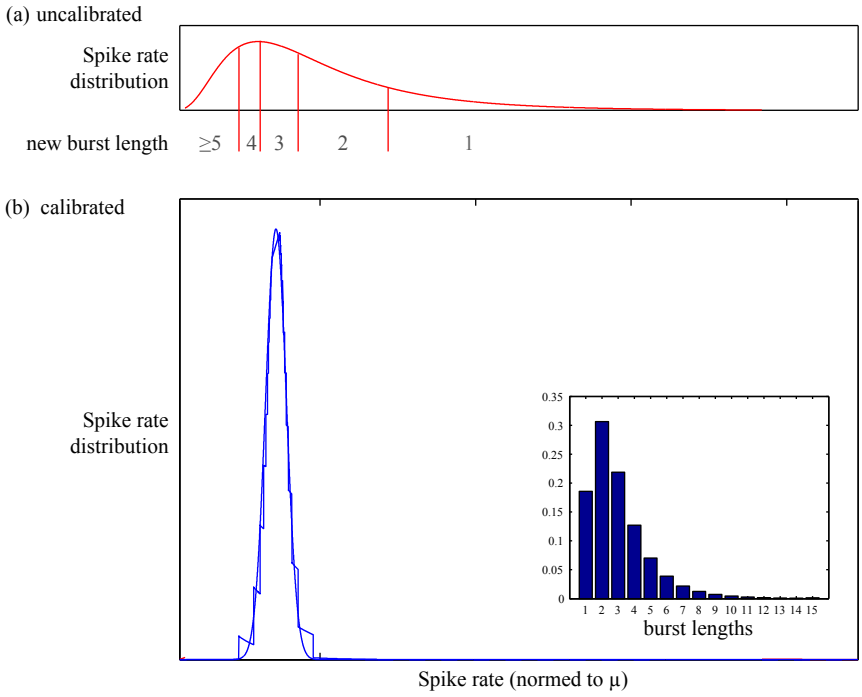

Figure 3.29: Procedure to estimate mismatch compensation using burst-length adaptation. (a) Distribution of uncalibrated firing rates. The firing rates were measured by initializing all burst lengths to 3. We do not use the measured distribution of a specific chip, but the fitted distribution that we obtained with the mismatch characterization because our routine assumes a continuous function for the distribution. The indicated new burst lengths show the burst length the algorithm assigns to a neuron whose firing rate falls in the indicated range. For example, a neuron whose firing rate is in the range indicated with 'new burst length 1', gets a new burst length of 1 assigned, instead of the burst length of 3 at which its firing rate was originally measured. It's firing rate after calibration will therefore be multiplied by 1/3, moving it closer to the mean of the distribution. In (b) we repeat this procedure for all neurons in the range indicated with '1', shifting the whole range of the distribution in (a) onto the mean. Similarly, the range indicated with 'new burst length 2' is shifted by multiplying it with 2/3, etc. for all burst lengths. The resulting distribution is fitted with a Gaussian distribution (smooth curve) to obtain the standard deviation of the calibrated firing rate distribution. X-axes: the firing rates were normed to the mean of the distribution in the calibrated case ($\mu = 1$). Y-axes: the distribution was normed so the area under the curve is 1. The same scale is used for (a) and (b). *(continued on next page)*

Figure 3.29 (*continued from last page*): Inlay: distribution of burst lengths after calibration. The algorithm optimizes the ranges of the burst lengths to obtain the distribution with the smallest variation, with the boundary condition that the average burst length in the calibrated case should be 3. If initial and average burst lengths are both set to 3, the distribution of the calibrated firing rates has about the same mean as in the uncalibrated case. For other initial or average burst lengths, the same resulting variation can be reached, but the mean of the distribution in the calibrated case varies. The shown example is so effective because the initial variation is large.

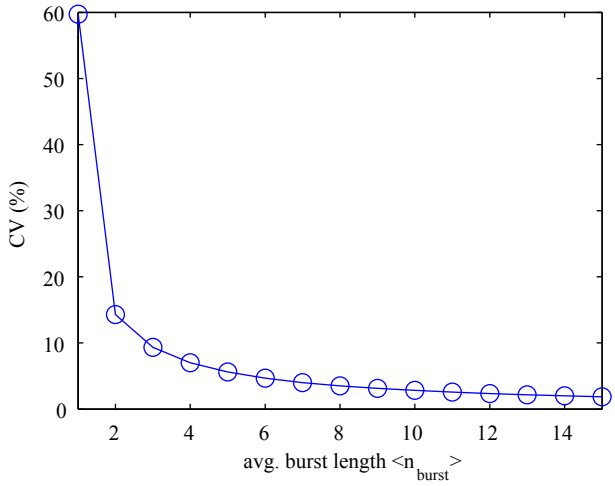

Figure 3.30: Minimal variation that can be achieved using burst length adaption versus the average burst length in the calibrated case, obtained with the optimization algorithm described in Figure3.29. The average burst length determines the bandwidth that is taken on the AER bus. Starting from an uncalibrated coefficient of variation of 60%, the mismatch drops significantly with only 2 spikes in a burst, but saturates at longer burst lengths.

Our D/A implementation is inspired by [Linares-Barranco et al., 2003, 2004], and was first implemented by Ying-Xue Wang on a test chip with 16 neurons [Wang and Liu, 2006]. Here we present data from this test chip. Later the D/A converters were also used on the third chip version of the CAVIAR object chip.

There is one D/A converter global for all neurons. The D/A converter generates the synaptic currents based on an external masterbias [Delbruck and Lichtsteiner, 2006]. The current is copied through a current mirror to all synapses.

We performed the same theoretical analysis for the mismatch compensation with D/A converters as for the burst-length adaptation, see Figure 3.31. Instead of the average burst length, the boundary condition of the optimization is the fixed range of D/A values.

Figure 3.32 shows the reduction in mismatch that can be achieved with D/A compensation of different resolution, assuming an initial coefficient of variation of 50%.

Again, the mismatch calibration is most efficient for systems with large variance. If the variance is smaller, not all D/A values can be used, but only a subset, see Figure 3.33.

The problem is that the D/A converters we considered until now span a range from zero to a maximum current $[0; I_{max}]$. While I_{max} can be adjusted through the master current to match the upper limit of the firing rate distribution, a large portion of the lower available D/A values falls into a range that is below the lowest spike rate, as illustrated in Figure 3.33. [Linares-Barranco et al., 2003] propose a D/A converter that operates in the range $[I_{max}/2; I_{max}]$. We use a bias current I_{bias} that is added to the D/A output. The total range of the D/A converter is then $[I_{bias}; I_{bias} + I_{master}]$. To determine I_{bias} and I_{master}, we first set I_{bias} to a current that is equivalent to the minimum firing rate in the distribution. I_{master} is then adjusted to be equivalent to the width of the distribution. The available D/A values cover the whole range of available firing rates, resulting in an optimal mismatch compensation.

The synaptic weights are not necessarily linear. We therefore measured the synaptic weight for each D/A value separately, see Figure 3.34.

To minimize the mismatch we selected the D/A values for each neuron so that the synaptic weights are as close together as possible. Figure 3.35 shows the improvement in the variation of coefficient for different synaptic weights. For a value of about 0.025 the resulting CV is minimal. For higher synaptic weights not all outliers can be corrected since their value is already close to limit of the range of D/A values. This results in a larger CV after compensation. We can understand this effect intuitively on Figure 3.34: Selecting the same synaptic weights is equivalent to the intersection of a horizontal line with the synaptic weights for each neuron (dashed line). If the target synaptic weight is chosen too high, the line will not intersect with all neurons, that is not all outliers can be corrected.

Using D/A converters to program the synaptic weight is a powerful approach.

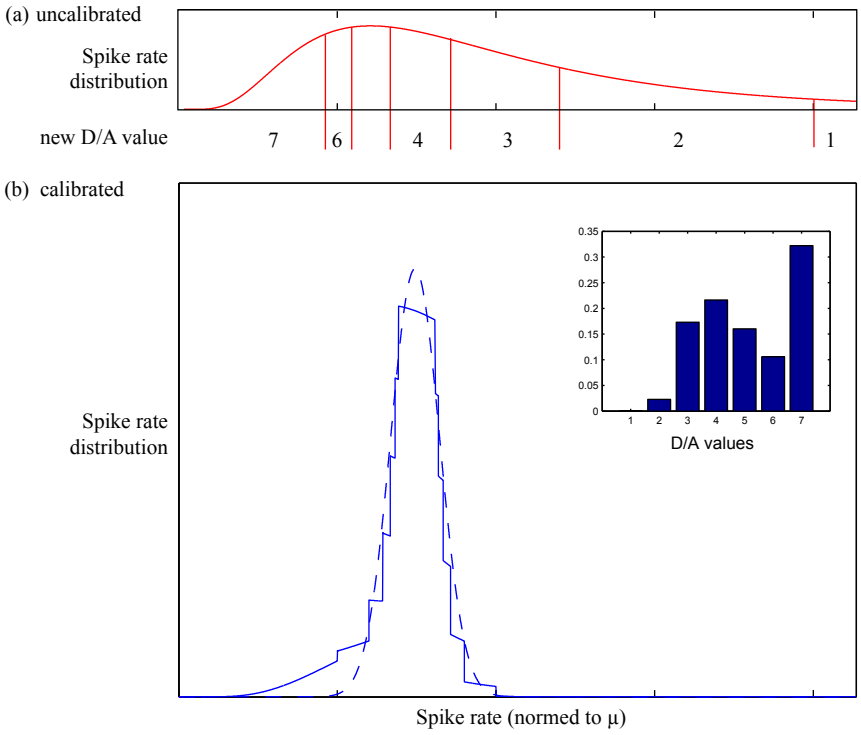

Figure 3.31: Procedure to estimate mismatch compensation using D/A converters. See Figure 3.29 for a description of the optimization algorithm, with D/A value instead of the burst length. Here we assume that the uncalibrated distribution was measured with the D/A converters initialized to a value of 4. The algorithm does not have to consider a boundary condition like the average burst length, since all D/A values can be chosen equally. If the initial D/A value is chosen in the middle of the range of D/A values, the distribution of the calibrated firing rates has about the same mean as in the uncalibrated case. The shown example is so effective because the initial variation is large.

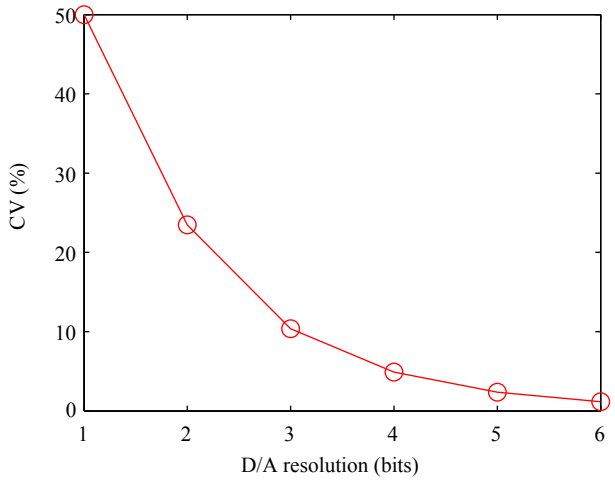

Figure 3.32: Mismatch compensation achieved by different resolutions of the D/A converter, starting from an uncalibrated coefficient of variation of 50%.

The synaptic weight is stored externally in cheap digital random-access memory (RAM). Compared to burst-length adaptation, it requires only a moderate amount of bandwidth: if we assume that the synaptic weight is transmitted with every spike (since in general in a neuronal network most of the synaptic weights will be different), the amount of data transmitted on the bus is doubled, at least for our two-spike communication scheme. In contrast to burst-length adaptation, D/A converters require special on-chip circuits. The main design challenge for the on-chip D/A converters is a short time until the weight settles at the synapse, so the handshaking on the bus is not delayed.

Stochastic Spike Transmission

The last procedure to influence the synaptic efficacy that we present here is based on stochastic spike transmission. In cortex, synaptic transmission is reported to be unreliable, for example see [Stevens, 1993]. Only a fraction of the spikes arriving at a synaptic bouton will trigger a response. In general, the synaptic weight is a product of the numbers of available vesicles in the bouton, the probability of release p, and the effect on the post-synaptic neuron.

Francisco Gomez Rodriguez implemented stochastic spike transmission on the USBAER mapper board. Together with the synaptic connection, the mapping table contains an entry for the probability of release. Every time a spike is received, a random number is generated and compared to the probability that is stored for this synaptic target. If the random number is lower than the

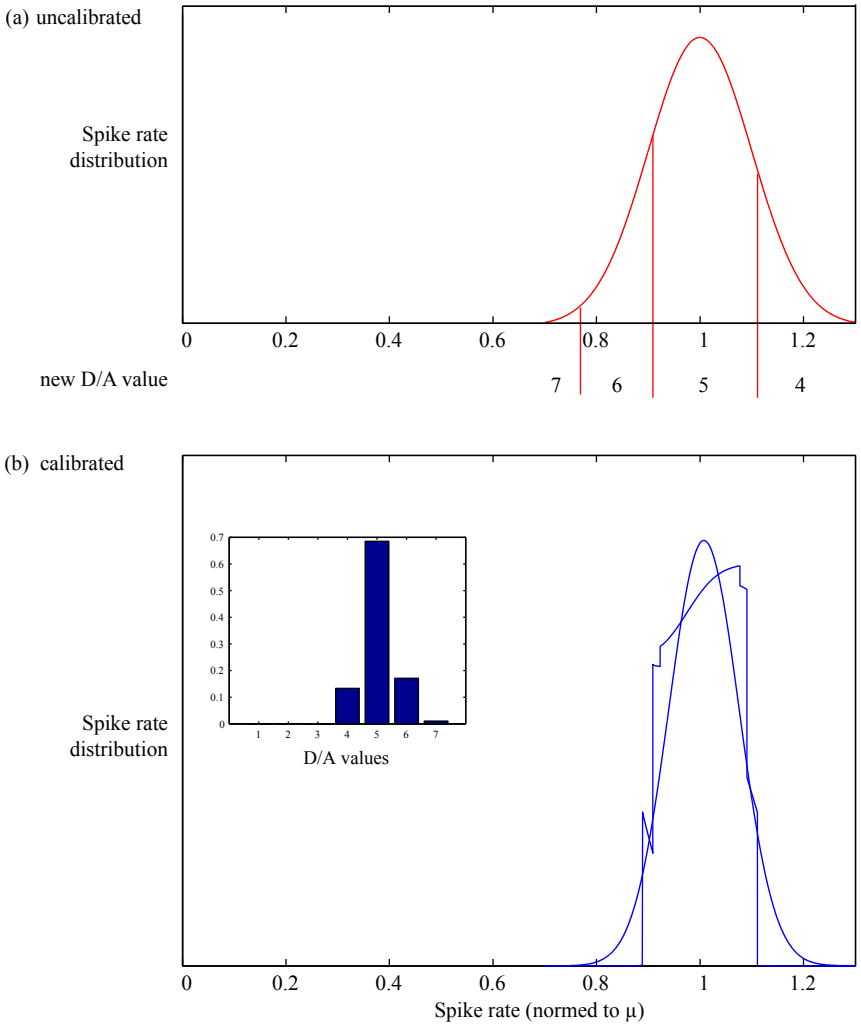

Figure 3.33: Procedure to estimate mismatch compensation using D/A converters. See Figure 3.29 and Figure 3.31 for a description of the optimization algorithm. Here we show that the mismatch compensation is less efficient if the initial variation in the network is small. Since the D/A converter considered here has a range of $[0; I_{max}]$, only three D/A values fall onto the distribution and can be used, see inlay.

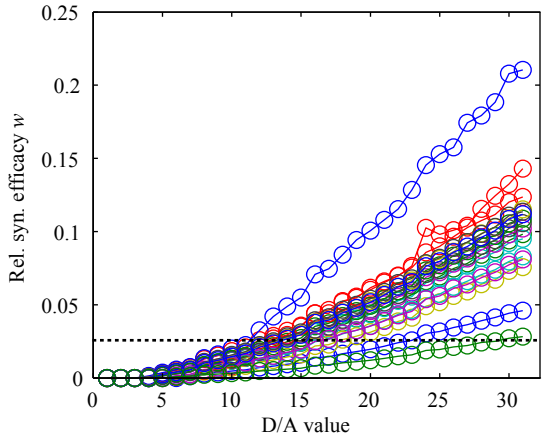

Figure 3.34: Measured relative synaptic weight for every D/A value and each of the 16 neurons of the test chip. The dashed line is the synaptic weight which results in a minimal coefficient of variation for the firing rates of the population. Data from D/A test chip, stimulation frequency is 1kHz.

stored probability, the spike is transmitted to the target synapse, otherwise it is discarded. The used 8 bits provide a resolution of 256 steps.

The model is appealing because it does not require additional on-chip circuitry. Due to the used digital implementation, the resolution is large and not subject to mismatch.

Introducing a probability of release might be a crucial limitation for some network models, especially for models based on single spikes or on regular rates, since the statistics of the spike rates are altered. If biologically plausible Poisson rates are used, the spike statistics are not altered, but just the firing rate is decreased. Stochastic spike transmission then provides an easy and efficient way of setting the synaptic weight and compensating mismatch, closely following the biological example.

Models that Exploit Mismatch

Although mismatch in our chips is due to the manufacturing process, variation in parameters must be present for biological synapses. The brain has found a way to compute with a large number of imprecise building blocks, presumably through learning and adaptation. Artificial systems consider mismatch almost always as a limiting factor, for example [Rodríguez-Vázquez et al., 2003]. However, mismatch can be useful for computation. The liquid-state machine [Maass et al., 2002] is one of the few models that exploit variation in the synaptic parameters for computation. In this model, a large pool of neurons forms a network

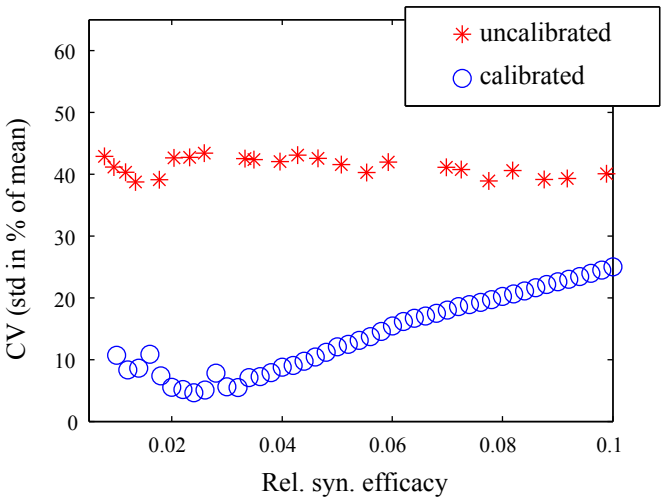

Figure 3.35: Mismatch compensation using D/A converters. The uncalibrated mismatch is about 40% (stars). With D/A converters, the mismatch can be reduced to less than 5% (circles). As described in the text, the achievable mismatch compensation is dependent on the chosen relative synaptic weight, with a minimum at about 0.025. For higher synaptic weights not all outliers can be corrected since their value is already close to limit of the range of D/A values, see text. Data from D/A test chip.

which connections are chosen randomly, as well as the synaptic parameters are chosen from a random distribution. The computational power results from the large amount of different functions that are formed by a network in which the building blocks show a large amount of variability.

The parameter distribution that these neural network applications consider is often assumed to be Gaussian. To what extent good can a Gaussian distribution be approximated by the skewed Gaussian that is intrinsic in our VLSI implementation? As shown in Figure 3.36, the distributions are fairly similar, especially if outliers with high firing rates are discarded. The Liquid-state machine could offer an elegant approach to exploit the intrinsic mismatch for computation. But in the liquid-state machine every synaptic connection is chosen from a distribution of parameters. In our implementation using AER, one synaptic circuit to multiplex all the synaptic connections onto one neuron. The synaptic efficacies are then distributed per neuron, not per synapse. An implementation would have to implement every synaptic connection as a separate synapse circuit.

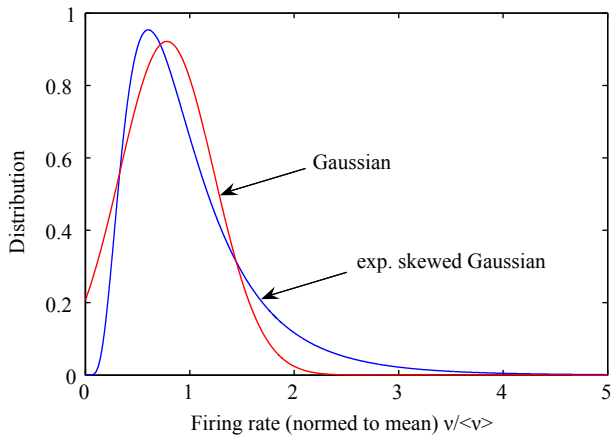

Figure 3.36: Approximation of a Gaussian distribution with an exponentially skewed Gaussian resulting from the intrinsic mismatch in the chip implementation. Many network models which consider synaptic variation assume the weights to be Gaussian distributed. Can the intrinsic mismatch of a chip implementation which follows a skewed Gaussian distribution be used for a model that assumes Gaussian distribution? The long tail of the high synaptic weights in the implementation can be discarded by eliminating those synapses. On the other side, the lower synaptic weights cannot be approximated since no neurons can be added to the population.

3.3.3 Winner-take-all Performance with Mismatch

Mismatch in the synaptic parameters is especially a problem in the uniform network used in the winner-take-all. Ideally, all synapses and neurons should behave the same for the same global parameters. Due to the intrinsic mismatch in the manufacturing process the synaptic efficacies show variation across the chip. On the first chip version, the mismatch was about 100%. With appropriate sizing of the critical transistors the mismatch was reduced to 20% on the second version. In addition the third chip version includes D/A converters to compensate for the mismatch. In this section we are interested in studying how the performance of the winner-take-all is affected by the mismatch.

Measurements in biological systems show a large variety of synaptic parameters. [Markram et al., 1998] report coefficients of variation of above 50% for dynamic synapses. Although this variation might seem huge, it is not necessarily significant for computation, since the measured synapses do not have to be part of the same function. For our winner-take-all network, all inputs contribute to the same function and the synaptic parameters should be as homogenous as possible. Variation in the parameters will directly affect the discrimination performance of the network.

Let us assume that all neurons receive input of the same regular frequency. Without winner-take-all connectivity, all neurons in the network spike with an output spike rate r_k. Due to mismatch in the synaptic efficacies, the spike rates differ slightly. We denote this vector of output spike rates with $r_{\max}$. The neuron with the largest synaptic efficacy has the highest output rate $r_{\max}$. With winner-take-all connectivity and the same input, the neuron with the highest firing rate wins the competition and suppresses all other neurons.

Now we assume that one neuron k will receive a stronger input than all other neurons. How much stronger has this input to be that the winner-take-all network will detect it? Neuron k will win the competition if its output rate is higher than that of the currently winning neuron which spiked with an output rate of $r_{\max}$. Since the synapses can be assumed to be linear, the required increase in the input is proportional to the difference in the output rates $r_{\max} - r_k$. The input has to be increased by

$$c_k = \frac{r_{\max} - r_k}{r_k} \cdot 100\% \tag{3.8}$$

Note that c_k depends on the maximum firing rate in the distribution, not directly on the variation in the firing rates. We can find an expected maximum dependent on the size of the population.

Be $g(r)$ the distribution of firing rates of the neuron population (or equivalently, the distribution of synaptic efficacies). We draw N samples from this distribution. For one sample, the expected value is obtained by integrating the rate r times the probability of this rate: $\int_{-\infty}^{\infty} g(r) r \, \mathrm{d}r$. We are not looking for the mean, but for the maximum. For one sample to be the maximum, all other

$N-1$ samples have to be smaller than its value r. The probability for this is

$$\left(\int_{-\infty}^{r} g(r)\mathrm{d}r\right)^{N-1} \tag{3.9}$$

Taken together we get

$$\langle r_{max}\rangle = N \int_{-\infty}^{\infty} \left(\int_{-\infty}^{r} g(r)\right)^{N-1} g(r)\, r\, \mathrm{d}r \tag{3.10}$$

Multiplying by N considers that each of the drawn samples can become the maximum. Figure 3.37 evaluates the maximum spiking rate for an increasing number of neurons in the population under the assumption that the synaptic efficacies are Gaussian distributed.

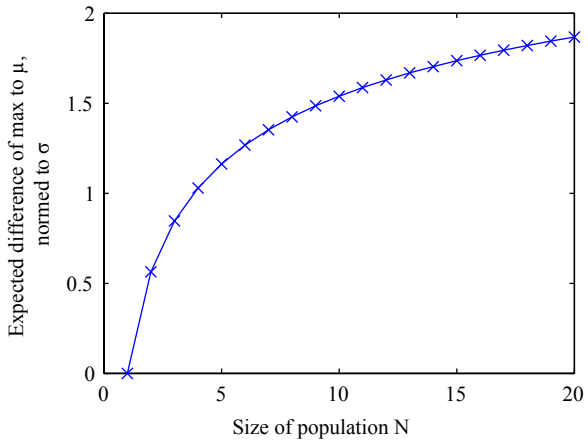

Figure 3.37: Expected maximum value versus population size N. We assume the firing rates $\mathbf{r}$ to be Gaussian distributed. Shown is the difference of the maximum firing rate $r_{\max}$ and the mean firing rate μ_r of the population, normed to the standard deviation of the distribution. For example, for $N = 9$, the expected maximum firing rate is $r_{\max} = \mu_r + 1.5\sigma$.

We now measure the factors c_k for the 64 neurons of the chip implementation. All neurons are stimulated with a spike train of a regular frequency of 100Hz. We set the parameters so that the number of input spikes needed for a neuron to reach threshold is nine. With mismatch, the vector of output spike rates $\mathbf{r}$ has a mean of $\mu_{\mathbf{r}} = 11.28\mathrm{Hz}$, that means on average a neuron needs 8.86 input spikes to reach threshold. In the scenario of uncalibrated mismatch, the standard variation of the output rates is $\sigma_{\mathbf{r}} = 11.69\mathrm{Hz}$ and the coefficient of variation

$CV = \sigma_{\mathbf{r}}/\mu_{\mathbf{r}} = 103.7\%$. With uniform input, the network selects the neuron with the highest effective excitatory weight due to mismatch as the winner, here $r_{\max} = 42.5$Hz. To select a different neuron k, its input frequency has to be increased by $c_k \cdot 100\text{H}z$. On average the increase is

$$\langle c_k \rangle_{\text{uncalibrated}} = \frac{r_{\max} - \mu_r}{\mu_r} \approx 277\% \tag{3.11}$$

We calibrated the network using burst length adaptation as described in Section 3.3.2. The resulting coefficient of variation for the calibrated rates is about 10%. We measured c_k for each neuron by increasing its input spike rate until it is selected as the winner at a frequency of f_k. The increase factor is then $c_k = (f_k/100\text{Hz}) - 1$. The measured c_k are listed in Figure 3.38. The neuron for which the spike rate does not have to be increased ($c_k = 0$) is the neuron that has the highest firing rate due to mismatch. The mean of the increase factors is

$$\langle c_k \rangle_{\text{calibrated}} \approx 10.2\% \tag{3.12}$$

which is about the same as the coefficient of variation of the spike rates themselves, that is without calibration.

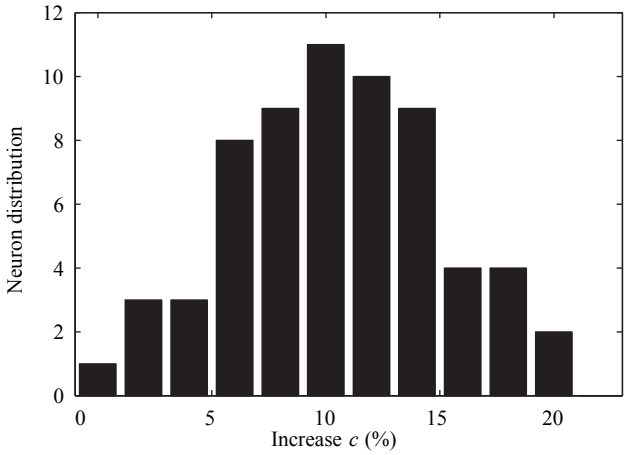

Figure 3.38: Discrimination capability of the winner-take-all with mismatch. Shown is the distribution of increase factors c_k by which the input spike rate of a neuron k has to be increased to select this neuron as the winner. The neuron with $c_k = 0$ is the neuron that has the highest firing rate due to the mismatch. Data from first chip version with mismatch calibration using burst length adaptation.

3.3.4 Discussion

In this section we characterized mismatch in analog VLSI implementations and discussed schemes to compensate for it. Mismatch affects the synaptic efficacies in two ways, with a multiplicative and an additive term (Equation 3.3). The multiplicative term is caused by imperfect matching of the transistors in the current mirror of the synapse. In effect, the synaptic efficacies (and the resulting neuron firing rates) follow a Gaussian distribution that is exponentially skewed (Figure 3.18). Since this mismatch affects the gain of the synapse, the coefficient of variation (CV) of the synaptic weights across the network is constant for different weights (Figures 3.19 and 3.22).

For small synaptic efficacies, variation is dominated by the leakage currents of the transistors in the soma circuits of the neurons. This mismatch has an additive effect since the input has to exceed the leakage currents to drive the neuron into a spiking regime, thus the standard deviation across the neurons of the network is constant. The variation of the leakage currents is large compared to the variation in the synaptic efficacies (Figure 3.19). We avoid the effect of leakage current variation by using large synaptic weights or high stimulation frequencies when measuring the synaptic efficacies.

We compared different schemes to set the synaptic efficacies and to compensate for the mismatch in the synaptic efficacies. Neuron sorting compensates for mismatch by selecting only the neurons whose synaptic efficacies are close together. The scheme is not efficient but easy to use if only a small part of the neurons on a chip are needed for an experiment. Burst length adaptation significantly reduces the mismatch for large variation in the uncalibrated case, and we used this technique extensively on the first version of the chip. Obtaining low variations in the synaptic efficacies would require too much bandwidth on the communication bus. This is solved using on-chip D/A converters that are programmed from external memory in the AER remapping. We presented measurements from a test chip [Wang and Liu, 2006] that show the mismatch compensation using D/A converters is quite effective. Another scheme, stochastic spike transmission, models an effect present in biological synapses. Implemented in the AER mapper, we can also use it to set the synaptic weight if the inputs are encoded as Poisson spike rates without considering single spike timing. Since our models depends on the timing of single spikes, we conclude that D/A converters are the most effective scheme to compensate mismatch in our network, and they were included in the third chip version.

In Section 3.2.1 we discussed how mismatch affects the performance of the network to discriminate between the input frequencies. Variation in the synaptic parameters limits the discrimination performance since the winning neuron has to exceed the highest spike rate of the case of uniform input. Here we demonstrated the discrimination performance in both the uncalibrated and the calibrated case for the first chip version. Without calibration, the spike rate of a neuron has to be increased by 277% to select this neuron as the winner, with

calibration only by 10%.

3.4 Embedding Software Learning in Spiking Systems

Much recent research in spiking neural networks focuses on the dynamic properties of network models such as learning algorithms based on synaptic plasticity and global reward signals, development of connectivity, and modulatory functions such as gain control. In this section we discuss a framework that allows the exploration of the dynamic properties of network models in real-time neural networks by combining hardware spiking neurons and software agents. Local analog and continuous-time computation is performed in the hardware, while *higher-level functionality* is implemented in software. By higher-level functionality we understand whatever algorithms are not implemented in the currently available hardware, for example learning algorithms based on synaptic plasticity and global reward signals, development of connectivity, and modulatory functions such as gain control. This new approach allows a wide variety of algorithms to be tested quickly and could enable real-time systems with large computational power to be assembled. The framework has been published in [Oster et al., 2005].

Several projects have focused on combining hardware based spiking neurons with dynamically reconfigurable connectivity. The Silicon Cortex (SCX) project [Deiss et al., 1999] proposed connecting multiple chips using spiking communication and already incorporated the possibility of building integrated hardware and software models of the kind we propose here, although no such models were actually implemented at that time due to the presence of a critical bug in the host communication channel. Similar systems are used by various groups. The IFAT system [Vogelstein et al., 2004, Mallik et al., 2005] is closest to our approach, however, we separate the hardware and software parts to achieve greater flexibility, higher performance and easier implementation of the algorithms, for example in MATLAB.

As described earlier, analog VLSI is used to implement models of biological neurons with transistor circuits that are integrated in large arrays on a chip. The connectivity between the neurons is implemented by the transmission of spikes over a multiplexed bus using the address-event representation (AER) protocol. Each spike is represented by the address of the source neuron or the receiving synapse and is transmitted asynchronously. A mapper translates the addresses of the sending neurons to lists of receiving synapse addresses using a look-up table, thus allowing for arbitrary intra- and inter-chip connectivity between neurons. Various networks and input sensors can be combined to form a real-time multi-chip system, see Figure 3.39. A monitor translates spikes from hardware to software, while a sequencer provides the reverse translation. The mapper, monitor and sequencer are integrated on a PCI-AER board [Dante and Del Giudice, 2001] which plugs into a PCI slot in a desktop computer.

The higher-level functions use software agents that are embedded in the

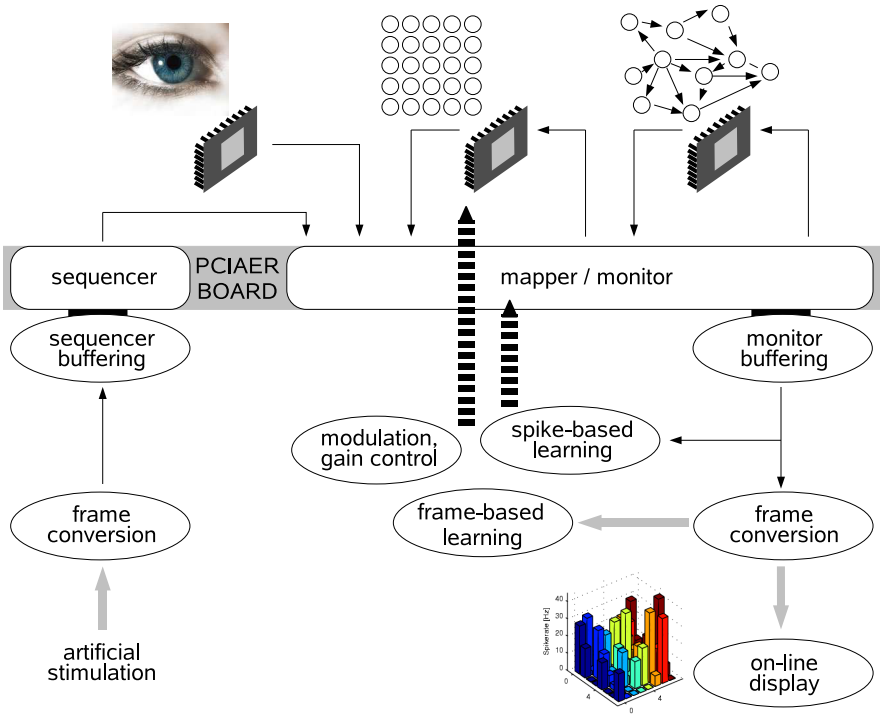

Figure 3.39: Overview of the system architecture. Real-time spiking neural networks are implemented in VLSI and integrated on a chip (top). As examples, a retina, a feedforward network and a recurrent network are shown. The neurons communicate using the address-event representation (AER) protocol (black arrows). A PCI-AER board monitors, sequences and remaps the spikes to implement the connectivity. Higher-level functions such as on-line analysis, learning algorithms, modulatory functions and artificial stimulation are implemented in C++ software agents (bottom). The agents transmit and receive spikes to and from the hardware using AER network packets and can change the connectivity and parameters of the network by modifying the mapping table and bias voltages (dashed arrows). Analysis agents transform the spike trains into a frame-based format which represents the activity of a neuron population in the chosen coding scheme (gray arrows). This allows agents implemented in slow environments such as MATLAB to be integrated into the framework. As an example, a 3D bar chart displaying the instantaneous firing rate is shown.

system. In this framework an agent is an independent software process that implements a particular higher-level algorithm. At present there are agents for spike train analysis, on-line display, learning and modulation functions and for stimulation. Multiple agents can run concurrently. Each agent communicates with the hardware neural network by receiving spike trains or activity from the network, and can change the synaptic connectivity and adjust the parameters of the neurons. Agents can also stimulate the network with artificial spike trains, providing input from parts of the system which are not implemented in hardware. Event-based agents, that are agents that perform computation based on single events, are implemented in C++, while agents that operate on the statistics of the activity of the network and do not require a low latency are implemented in MATLAB.

In the software, a spike is represented by a data structure containing its address and a timestamp recorded when the spike is captured. The timestamp is required to preserve timing information when the spikes are buffered. The monitor agent sends blocks of spikes including their timestamps as network packets to receiving agents. We chose UDP for this software spiking communication because it is fast and allows several agents to receive the spike trains at the same time using multicasting. UDP, the 'user datagram protocol' [Postel, 1980b] is an unsecured, unidirectional protocol for the transmission of network packets. We use network protocols to transmit data between agents because their implementation is highly optimized in current operating systems and independent from the operating system itself, in contrast to alternative transfer mechanism as first-in-first-out queues (FIFO) or shared memory buffers. In addition the receiving agents can run on the same computer as the sender or on different machines.

For many applications, we are not interested in the spike train itself, but rather in the statistics of the activity of the neurons in the network. Depending on the chosen coding scheme, this can be the instantaneous spike rate, the spike count, the time to the first spike, or any other suitable measure. Analysis agents transform the spike train into one of these activity-based representations. We implement these analyses in independent agents because they are often needed by several agents further on in the processing. In section 3.4 we will describe an algorithm that estimates the instantaneous spike rate with a running average.

An agent further along the processing chain requests this activity estimation. To do so, it first specifies the list of addresses of the neurons to be monitored. The analysis agent then transmits the activities of these neurons as a vector. We call this 'frame-based representation'. In contrast to conventional frame-based representations, the timing is asynchronous since the frame can be requested at any time and the analysis agent update their activity estimation with incoming spikes. Frames are transmitted between the agents using the Transmission Control Protocol (TCP) which provides a reliable, bidirectional, point-to-point communication channel [Postel, 1980a].

The frame-based representation makes it possible to include agents in the

framework that have long response times. Typical examples are parsed environments such as MATLAB. As an example, an on-line display agent can request frames and display them with a fixed refresh rate independently of the amount of spikes received.

The framework allows a variety of learning and modulation algorithms to be explored by implementing them as agents. The agents can be either event-based or frame-based. As an example of an event-driven agent, we implemented an agent that uses spike-time-dependent plasticity (STDP) [Abbott and Nelson, 2000]. The agent is configured with the addresses of the post-synaptic neurons and their pre-synaptic afferents. All incoming spikes are buffered and the agent checks whether a post-synaptic neuron spiked. If so, the buffer is scanned for spikes from pre-synaptic neurons that fall within the time window around the post-synaptic spike and long-term depression or potentiation is calculated. The synaptic efficacy is changed on the fly in the mapper's look-up table using burst length variation (see section 3.3.2). The performance of this implementation is listed in Table 3.40. Exploring STDP with this software-based approach has advantages over a hardware implementation in that the implementation time is shorter and testing is easier, since no new hardware has to be added on chip and all of the algorithm's variables are accessible.

All agents including the analysis agents are derived from a common C++ base class that encapsulates the spike input over UDP and a TCP connection that is used for frame requests and for configuring the agent. New agents can be quickly implemented into the framework by overriding just two functions in a derived class.

Table 3.40 shows the performance of the framework. We show both maximal throughput for standalone agents and throughput for a typical setup using an agent implementing STDP learning and a display agent. The main limitation on the hardware side is the transfer of data (spikes and synaptic weight updates) over the PCI bus. The driver for the PCI-AER board does not yet support interrupts, and the current board does not support bus mastering (a PCI bus mode that allows the fast transmission of large data blocks). Even with these limitations, the measured throughput is sufficient for many experiments because it refers to the continuous spike rate, whereas biologically plausible networks typically exhibit bursts of spikes with high-frequency, and the average spike rate remains well below the maximum throughput.

With its modular architecture, our framework supports multiple agents using event or activity based computation. Software spike trains are broadcast to multiple receivers and statistics relating to different spike coding schemes can be requested in a frame-based representation. Thanks to the use of standard network protocols, the system is scalable and can be distributed across several computers. In a more complex setup, multiple agents are active, either during different developmental stages of the network, or to perform various learning and modulation functions in different layers of the network.

New hardware interfaces that implement the functionality of the PCI-AER

Maximum throughput (standalone agent)			
	rate s^{-1})	avg.(max.) latency	CPU load [%]
AER communication (up to 4 chips)	1.2M Spikes	1.2μs	-
Monitoring:			
scheduled, local	130 kSpikes	15 (90) ms	<5
scheduled, remote	94 kSpikes	310 (350) ms	-
busy poll	312 kSpikes	24 (63) ms	90
Synaptic efficacy updates	129 kUpdates	-	98
Typical setup (multiple agents)			
Monitor agent	53 kSpikes	15 (90) ms	8
STDP agent:			35
spikes of postsynaptic neurons	24 kSpikes		
Synaptic efficacy updates	16 kUpdates	44 (220) ms	
Spike-rate to frame conversion	53 kSpikes	25 (120) ms	3
On-line display (MATLAB/X)	2.3	-	24

Figure 3.40: Performance measurements. All rates given are maximal rates at which no or very few spikes are lost (<1 packet in 1s). The latency gives the mean (maximum) latency from a spike being recorded by the PCI-AER board until it is received and processed by an agent. All measurements were done on a standard PC (2.4GHz Pentium IV, Linux kernel 2.4.26). We tested several timing schemes for the monitoring agent. Scheduled monitoring refers to a mode where the agent polls the status of the hardware FIFO with a low frequency and transfers data if the FIFO is half full. This leaves enough computation time for other processes in between the monitor polls. Busy polling results in higher data rates, but takes up the complete processing time on the CPU. In local mode, spike packets are transmitted to agents on the same machine, in 'remote' they are distributed to other computers over the network.

board in single hardware modules have been developed as part of the CAVIAR project. These hardware modules can be inserted wherever needed into the data flow of the system. They also support much higher spike rates than the current PCI-AER board, of up to 25MSpikes/s [Serrano-Gotarredona et al., 2005]. Since they transfer their data to the computer over the Universal Serial Bus (USB) [Consortium, 2000], they can be used in portable applications where no PCI slot is available. Until now, only the USBAER framegrabber module is integrated in the framework, while others, for example the mapper, have been tested.

The framework we presented here can be used to quickly explore higher-level functionality in a real-time system. Through the use of software agents, it provides a rapid prototyping tool to test learning and modulation algorithms in a real-time system.

Online Spike-rate Estimation Algorithm

We describe our agent that estimates the instantaneous firing rate because we are not aware of an existing description of this algorithm. The algorithm estimates the instantaneous spike rate on-line, that is with a running average. It is equivalent to smoothing the firing rate measurements with a causal exponential filter, see [Dayan and Abbott, 2001, pp 13.], but in our version we formulate the algorithm iteratively, that is the estimation is updated at every incoming spike. See Figure 3.41 for an illustration.

Estimating the spike rate takes place when a new spike is received at time t. Let Δ be the current inter-spike interval. The spike rate is estimated with

$$\hat{f}(t) = \alpha^{(\Delta f_\alpha)} \hat{f}(t-\Delta) + \left(1 - \alpha^{(\Delta f_\alpha)}\right) \frac{1}{\Delta} \tag{3.13}$$

with $\hat{f}(t)$ the current spike rate estimation, $\hat{f}(t-\Delta)$ the spike rate estimation at the time of the last spike. α and f_α adjust the time constant of the exponential smoothing.

Let us assume we read out the current spike rate estimation at time t'. We define the access time t_a:

$$t_a = t' - t_{\text{last}} - \beta \cdot \Delta \tag{3.14}$$

with t_{last} the time of the last spike received and β a factor that is adjusted to estimated the change in the spike rates. The spike rate $\hat{g}$ that is read out is then calculated as

$$\hat{g}(t') = \begin{cases} \hat{f}(t_{\text{last}}) & : \quad t_a < 0 \\ \hat{f}(t_{\text{last}}) \cdot \exp(t_a/\tau) & : \quad t_a > 0 \end{cases} \tag{3.15}$$

with τ the time constant of an expected decrease in the spike rates.

parameter	value
α	0.99
f_α	10Hz
β	1.5
τ	0.5s
$\hat{g}_{\min}$	0.5Hz

Table 3.6: Parameters used in the online spike rate estimation.

We define a cut-off threshold $\hat{g}_{\min}$ under which the read spike rate is set to 0:

$$\hat{g} = 0 \quad \text{if} \quad \hat{g} < \hat{g}_{\min} \tag{3.16}$$

We empirically obtained parameters that result in a smooth graphical display of the firing rates present in our aVLSI system, see Table 3.6.

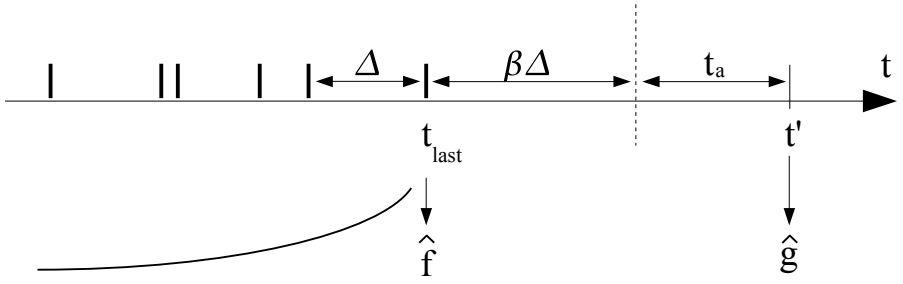

Figure 3.41: Algorithm to estimate the instantaneous spike rate of a neuron, with event-based estimation and request-based read-out. At every incoming spike, the estimation of the current spike rate $\hat{f}$ is updated, by weighting the last estimation and the current inter-spike-interval Δ. The weighting factors are corrected for the asynchronous update times, resulting in a smoothed rate measure with an exponential filter. When a readout of the current estimation is requested ($\hat{g}$), the algorithm first checks if the estimation of the spike rate is still valid. We assume that the firing rate did not change and the estimation is still valid if the last spike happened less than β times the last inter-spike-interval before the readout (dashed line). If more time has passed since the last spike ($t_a > 0$), we 'age' our estimation by weighting it with an exponential function.

Our online spike rate algorithm is efficient since the computation is only performed on incoming spikes, and the readout is independent from the estimation. We use it in the analysis agent that estimates the current spike rates for all neurons. Even for large populations of neurons no significant CPU time was used. In contrast to counting spikes in a time window, the algorithm does not

have to store a large amount of input spikes and is independent from problems with the window length that might cause buffer overruns. The USBAER boards in the frame grabber mode use a time window to count spikes. Since the time window is global, the resolution is limited: while neurons with high firing rates will cause the counter to overflow, neurons in the same population with low firing rates will only send a small numbers of spikes in the time window, that means that their spike rates are represented with only a small resolution (in more advanced schemes the count values are smoothed at readout).

Chapter 4

Application to Vision - Caviar

In this chapter we discuss the application of theory and implementation: embedding the winner-take-all network into a multi-chip spike-based vision system. The CAVIAR (Convolution AER Vision Architecture for Real-time processing) project is a multi-lab EU-funded research project[1] that explores spike-based processing in a multi-chip vision architecture. The winner-take-all is one of the building blocks and performs a decision on the most probable location of a moving object that is presented to the retina.

We will first discuss the biological inspiration for the CAVIAR architecture and its closest counterpart in modeling, the HMAX network (Section 4.1). We then describe the CAVIAR building blocks (Section 4.2 and the experiments in which the complete system is assembled (Section 4.3). The analysis of the data from the system is the main part of the chapter (Section 4.4). We characterize the input to the winner-take-all, fitting the model of a traveling wave of Gaussian shape that we developed in Section 2.3.2. The output, the predicted position of the object, is used to estimate the performance of the winner-take-all implementation in a large-scale system.

Figure 4.1 provides an overview of the CAVIAR architecture.

4.1 Comparison with the HMAX Network

Vision is a classical application of artificial intelligence, and many systems for visual processing have been developed with various degree of biological plausibility. One recent model is the HMAX network [Riesenhuber and Poggio, 1999]

[1] CAVIAR project is part of the FET (Future and Emergent Technologies) initiative, EU 6^{th} framework, user-friendly information society program, sub-program life-like perception systems. IST-2001-34124.

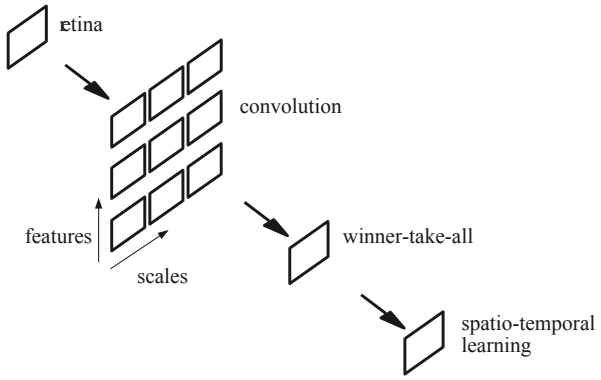

Figure 4.1: Overview of CAVIAR architecture. The different functions are separated into individual building blocks. The retina as the input sensor performs temporal processing by detecting contrast changes and encoding them as ON/OFF spikes. The convolution stage performs spatial processing, by convolving the input spikes with pre-programmed kernels. The kernels can detect different features or different scales, depending on the programming and the available number of convolution chips. The winner-take-all converges this information to a definite object position. The spatio-learning consists of a delay line chip, which expands the object position over time, and an unsupervised Hebbian learning chip that classifies the object trajectories in time. All signals between the stages are encoded and transmitted as spikes.

based on the 'Neocognitron' by [Fukushima, 1980]. It is based on the experimental findings that neurons in visual cortex show an increasing amount of specificity and invariance from lower to higher visual areas. For example, cells in the early visual areas respond strongly to oriented stimuli, while cells in the higher visual areas respond to views of complex objects such as faces [Ungerleider and Haxby, 1994].

Similarly, simple and complex cells in V1 or cat striate cortex show different degrees of invariance: both respond strongly to oriented bars, but whereas simple cells in general have small receptive fields with a strong phase dependence, complex cells have larger receptive fields and no phase dependence [Hubel and Wiesel, 1962]. This observation led Hubel and Wiesel to propose a model in which simple cells with neighboring receptive fields feed into the same complex cell, resulting in a phase-invariant response. The idea of simple and complex cells was generalized to obtain different types of invariance (translation, rotation, scale and rotation in three dimensional space), as well as to describe cells that respond to more complex stimuli. For this purpose, pairs of layers with simple and complex cells are alternated. The simple cells of the next higher-order layer receive input from the complex cells of lower complexity [Fukushima, 1980, Riesenhuber and Poggio, 1999]. The authors of the HMAX network point out that specificity (simple cells) and invariance (complex cells) requires different type of connectivities between the layers: simple cells build up specificity by processing their input with a weighted sum, as commonly used in artificial neural networks. Complex cells build up invariance by performing a MAX operation on their afferents, as in a winner-take-all circuit. Alternating these two processing principles leads to detection of quite complex objects under several instances of invariance. In the HMAX network this is shown for the example of paper clips that can be recognized invariant to different aspects.

The Caviar system adopts the principle of alternating the two types of connections for the simple and complex cells: the weighted sum (simple cells) is equivalent to multiplying the input channels with a kernel. As in HMAX, retinotopic arranged simple cells repeat this operation over the full input space, which is equivalent to a convolution operation. In CAVIAR, the convolution chip performs a spike-based convolution operation with a freely programmable kernel. Several convolution chips can be run in parallel with different kernels to build up a population of retinotopic S1 cells that are sensitive to different input patterns.

The winner-take-all operation is equivalent to the MAX function as performed by the complex cells. The 'object' chips implement winner-take-all networks in their input range. Since the chips are tiled into 4 quadrants, and several chips can be run in parallel, multiple MAX operations can be performed in parallel.

To understand the detailed computations performed in the HMAX network, we reprogrammed the network with access to examine the internal variables. See Figure 4.2 for an overview of the layers.

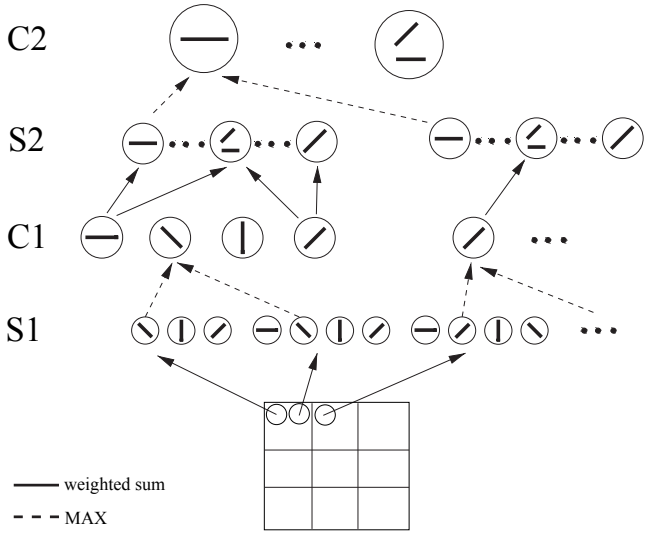

Figure 4.2: HMAX network. The network consists of 4 layers in which convolution operations (weighted sums) alternate with MAX operations. Cells in the first simple layer S1 extract oriented edges from the input image (bottom) by convolution with Gaussian kernels. Neurons in the first complex layer C1 pool spatially over a subregion in S1 with the MAX operation. In the second simple layers S2 cells generate every possible combination of the four orientations per C1 cell. For example, an S2 cell that combines C2 cells that are selective for an horizontal edge and an edge of 45° detects a corner at which these orientations are present. In the S1, C1 and S2 layers the computation is performed in parallel for different sizes, the so-called 'scalebands'. In the last layer, the second complex layer C2, neurons pool over all positions of the input image and all scalebands, resulting in a vector of the 4^4 combinations of orientations. Figure adapted from [Riesenhuber and Poggio, 1999, Figure 2].

S1 The first simple layer is a filterbank with filters of different sizes and orientations. The convolution kernels are two-dimensional second derivatives of Gaussians, ranging in size from 7x7 to 29x29 in steps of 2 pixels. After the convolution operation, the filter outputs are normalized to the sum of the squared pixel values. The filters are grouped into scalebands according to their size (sizes 7,9 form one scaleband, then sizes 11,13,15 etc.). A MAX operation is performed to get the strongest response within each scaleband for each pixel. The output of the S1 layer has the dimensions $n_{orientations} \times n_{scalebands} \times x \times y$ (the output sizes x, y vary with the filter size in the scalebands).

C1 The cells of the first complex layer pool spatially over a subregion with the MAX operation. The region of the pooling is varied with the size of the scaleband (for example pooling over 4 neighboring pixels for the smallest scaleband, then 6 pixels etc.). The output of C1 has the dimensions $n_{orientations} \times n_{scalebands} \times x \times y$, with x, y smaller than that of S1.

S2 The second simple layer calculates every combination of the four orientations for every pixel. If for example only a horizontally oriented line was visible at this image patch, the combination of only horizontal oriented filters will be strongest, equivalent to multiplying the horizontal filter value by four. At a corner, several orientations will be strong, and the combination will be high that contains all these orientations (for example, a corner with horizontal and vertical edge). With four orientations, every pixel is expanded to $4^4 = 256$ combinations, resulting in $n_{combinations} \times n_{scalebands} \times x \times y$ dimensions.

C2 Pooling in the second complex layer is extensively: for each combination, all pixels of the image in one scaleband, and then all scalebands are pooled in a MAX operation. The resulting output is a vector of 4^4 combinations of orientations.

We checked that our reprogramming generates exactly the same output as the original HMAX implementation.

While the different pooling steps and the processing in scalebands are motivated by the hierarchical processing in biology, the combination of oriented edges at the S2 layer seems arbitrary. Detecting corners of different orientations is useful for the dataset the HMAX network was designed for, that are images of three-dimensional paper clips in different rotations. The performance of the network for a real-world task such as face recognition is reported to be 'rather poorly' [Serre et al., 2002]. In this case, the performance can be significantly improved by incorporating learning [Serre et al., 2002, Louie, 2003]. Learning takes place at the connections C1-S2. Neurons in the S2 layer do not represent simple combinations of orientations, but detect patterns significant for the input images. This leads to a significant improve in performance, yielding the network performance comparable to artificial intelligence systems [Serre et al., 2002].

Another criticism is the high amount of resources needed for the HMAX network. An explicit representation of the output of the S2 stage is very demanding

in memory, requiring a vector of over 1000 entries per filtered pixel. The HMAX network solves this problem by combining filtering and pooling operation, so no intermediate storage of the S2 representation is necessary. In a biological neural network an explicit representation would be necessary.

The output of the network is classified by units that are tuned to specific views of objects or object components (VTUs). Expansions of the HMAX network pool over these VTUs, yielding object-tuned units, similar to neurons found in the anterior inferior temporal cortex (AIT), and task-related units, similar to neurons found in inferior temporal cortex (IT) or pre-frontal areas (PFC) [Riesenhuber and Poggio, 2000, 2002].

With its high demand on resources, the CAVIAR system is not capable of duplicating the HMAX network. The CAVIAR architecture does not repeat the combination of convolution and MAX operation like HMAX does (S1/C1, S2/C2), but restricts itself to one convolution and one winner-take-all layer. The HMAX network shows that for a vision application the combination of convolution and MAX operation is useful and can be repeated in a hierarchy as a general building block. Since CAVIAR has only one 'simple' and one 'complex' layer, the basic features need to be more complex so that the network can solve real-world tasks. The bank of convolution chips with its freely programmable kernels offers the possibility to incorporate kernels of arbitrary two-dimensional shape. Various theoretical frameworks to choose such kernels are available, for example see [Ullman et al., 2002] for an approach based on mutual information.

With only one pair of simple and complex layers the system is not able to obtain invariance against several aspects of vision, such as translation, rotation, scale and three-dimensional rotation. In the current version, CAVIAR achieves translation-invariance since the convolution chip performs convolution on the full input image. Scale-invariance can be achieved to a certain extent by programming several convolution chips with kernels of different size. Rotation invariance, invariance against rotation in 3D-space, or invariance against small changes in the perceived shape can only be achieved by increasing the number of convolution chips to incorporate every possible view of an object. Here the hierarchical organization of HMAX provides a clear advantage, although the mechanisms by which the network achieves these invariances are still unclear.

Some of the missing invariances could be compensated with an active vision system. In such a system the next focus of attention is selected by feature detectors of relatively low level of complexity, but translation-invariant. After a saccade to a point of interest, feature detectors of high selectivity recognize an object. The complex feature detectors do not have to be translation-invariant since the object is centered on the fovea.

CAVIAR can be extended in several ways. First, additional pairs of layers performing a weighted sum and a MAX operation could be added. As in the HMAX network, these layers do not perform retinotopic convolution operations but require a weighted sum that is specific per pixel. This could be implemented by programming the connectivity into the mapping tables, and by using

programmable synaptic weights for these connections. Such an extension could already be implemented in the current architecture, if the 1-1 connectivity between the output of convolution chip and the input of the winner-take-all is replaced by more complex synaptic connectivity.

HMAX reports an increase in performance if the features are learned. In section 3.4 we described a framework that incorporates learning algorithms of arbitrary complexity into spiking hardware such as CAVIAR.

4.2 Building blocks

One of the design goals of CAVIAR is to separate all functional modules into individual building blocks. Each building block is described by one specific function: the detection of temporal contrast edges for the retina; spike-based convolution with programmable kernels in the case of the convolution chip; the maximum operation of the winner-take-all network on the 'object' chip; and the delay line and classifier chips to learn spatio-temporal patterns in an unsupervised spike-based learning algorithm.

We describe the modules only to the level of detail that is needed for an understanding of the complete system, further details can be found in [Serrano-Gotarredona et al., 2005] and the publications listed at the building blocks. Table 4.1 gives an overview of the available modules and their communication speed.

Building block	**Function**	**Size (pixels/neurons)**
Retina	detects temporal contrast edges	128^2x2 (on/off)
Convolution Chip	programmable 32x32 convolution kernel	4x32^2 (tiled / parallel)
'Object' Chip	multi-dimensional spike-based winner-take-all	32^2 or 4x16^2 (4 synapses each)
Delay line	programmable delays	880 elements
Learning chip	associative Hebbian learning	32 (64 synapses each)
Interfaces	connectivity; monitoring and injecting spike trains	up to 2^{16} addresses

Table 4.1: Overview of CAVIAR building blocks.

Retina The design of the CAVIAR retina is based on a previous design by Jörg Kramer [Kramer, 2002], and is designed by Patrick Lichtsteiner and Tobi Delbruck at the Institute of Neuroinformatics (INI), Zurich [Lichtsteiner et al., 2006]. It detects temporal changes in the contrast. Contrast is the change in intensity normed by the intensity ($\Delta I/I$). The retina generates spikes if the contrast changes exceed a threshold, for example one spike encodes a contrast change of 10%. Positive changes in intensity, that are transitions from dark to bright, are encoded as 'ON' spikes, negative changes as 'OFF' spikes (see

Figure 4.3). Every pixel adapts to the local intensity, so for still images no events are generated. With the local spike generation and the asynchronous communication with the external world the retina is significantly different from conventional frame-based imagers. For example, small but fast moving objects can be captured with high speed.

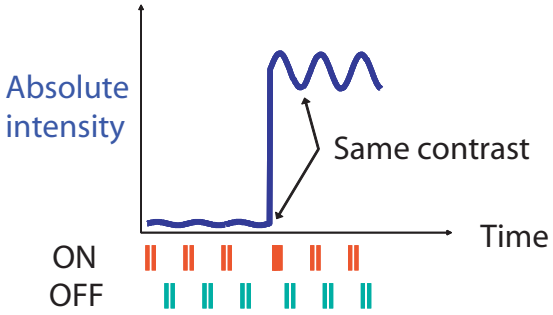

Figure 4.3: Principle of retina pixel operation. Contrast is the change in intensity normed by the absolute intensity. The retina generates spikes per temporal contrast change. Positive changes in intensity are represented as 'ON' spikes, negative changes as 'OFF' spikes. The shown stimulus is a sine-wave that is modulated with the same contrast independent of the absolute intensity. The retina responds with the same pattern of ON and OFF spikes at high and low levels of absolute intensity. Figure by Tobi Delbruck, reproduced with permission of author.

The retina has been optimized for usability and reliable operation. Compared to earlier imagers, the resolution has been increased to 128x128 (times two to count for ON and OFF spikes), and the mismatch has been reduced to 8%. Integration and miniaturization of the interface logic has led to a compact sensor if the retina is operated as a standalone module. Peak spike rate of the retina is up to 1Meps, although real-world stimuli normally result in much lower rates. Further details are published in [Lichtsteiner et al., 2006].

Convolution Chip The convolution chip was designed by Rafael Serrano-Gotaredona and Bernabe Linares-Barranco at the Institute of Micoelectronics (IMSE), Sevilla [Serrano-Gotarredona et al., 2006]. It performs a two-dimensional spike-based convolution. In conventional image processing, convolution normally involves processing a kernel matrix at all positions of the input image. At each position, the output is the sum of the element-wise multiplication of kernel and underlying input image. Spike-based convolution reverses this procedure: For each incoming spike, a kernel is copied onto the output matrix, centered on the pixel the spike is addressing. The output matrix consists of neurons that integrate the kernel weights on their membrane capacitance. Each

neuron sums the input that is generated from processing the kernel for incoming spikes at neighboring locations of the neuron. When the sum in the membrane potential exceeds a threshold, an output spike is generated. The computation is only based on local information and performed in continuous time. If the input is rate-coded, the output of the spike-based convolution is equivalent to a conventional convolution operation with thresholding.

Since the kernel weights can be positive and negative, the integration can also result in a negative value. The convolution chip implements a neuron as a symmetric integrator with two ranges of values. Resting and reset potential are not at ground, but at an intermediate value, so integration can be performed towards higher voltages for positive weights and towards lower voltages for negative weights. Consequently, two thresholds are implemented at the end of the integration range that generate 'positive' and 'negative' spikes. A forgetting mechanism drives the neuron back to the resting potential so the convolution operation can be adjusted for time-varying input.

The convolution chip implements an array of 32x32 neurons (each with positive and negative integration). The kernel can be programmed arbitrarily with a resolution of 4 bits up to a size of 32x32. Since each pixel considers spikes in its surrounding as defined by the kernel, the chip processes input spikes from the area of the 32x32 neuron array plus the size of the kernel. Multiple convolution chips can be tiled to process a larger input space, for example 64x64 pixels.

Detailed characterization and calibration of mismatch has been performed on the convolution chip using on-chip D/A converters, resulting in a standard deviation of the synaptic weights of 2%. The spike-based convolution operation allows for very fast detection of input patterns compared to conventional frame-based approaches. Input spike rates are 3-33Meps depending on the kernel size (see [Serrano-Gotarredona et al., 2006] for details).

'Object' Chip The object chip, whose layout was designed by Shih-Chii Liu at the Institute of Neuroinformatics, Zurich, reduces the dimensionality in the input space by determining the most probable location of an object. We implemented a cascade of winner-take-all networks on chip. On the first level, the strongest input is determined within each feature map. On the second level, the winners of each feature map compete to determine the strongest map. The resolution of this chip is a total of 32x32 neurons, with 4 winner-take-all networks of 16x16 operating in parallel on the first level. We described the implementation in detail in Chapter 3, see also [Liu and Oster, 2006].

Delay line and Learning Classifier This module was designed by Philipp Häfliger, Dep. of Informatics, University of Oslo (UIO) [Riis and Hafliger, 2004]. It learns to classify spatio-temporal patterns using unsupervised spike-based Hebbian learning. First, the input is expanded over time. The delay line chip contains 880 delay stages that can each hold 8 spikes. Spikes can be inserted at any point of the queue, and each delay stage can be programmed to output the

spike, or transmit it to the next delay stage. With this configuration, a variety of delays can be programmed. The output is a spatial representation of time in terms of address space, see Figure 4.4, left.

The classifier implements a network of 32 neurons with 64 synapses each, see Figure 4.4, right. The synapses use a learning algorithm with spike-time-dependent plasticity. Multi-level weak analog memory stores the synaptic weights. Global winner-take-all inhibition ensures that only one neuron has active output at a time. This also serves to decorrelate the learning, since the winner-take-all mechanism prevents two neurons to learn the same input pattern. Initially every neuron is biased towards a different input pattern due to mismatch. The neurons then learn to represent the complete input space that is presented. At the same time (or when learning is stopped), the network behaves as a classifier since the winner-take-all selects only the neuron with the strongest input. If delay line and the classifier chip are combined, the module can learn and classify spatio-temporal patterns. The delay line chip expands the temporal input pattern into address space and the classifier learns this representation. The delays have to be chosen so that the temporal dynamics of the stimulus are represented. For details see [Riis and Hafliger, 2004].

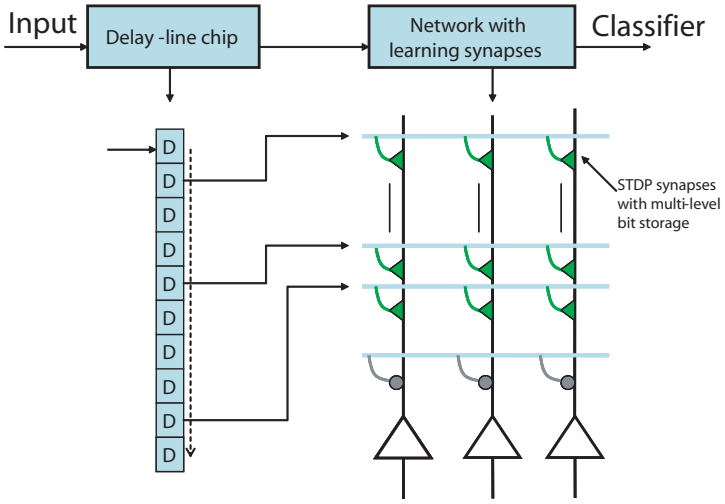

Figure 4.4: Learning of spatio-temporal patterns. Input spikes are expanded over time with a delay line into address space. The synapses of the classifier (triangles) implement STDP learning with multistable analog memory. Global winner-take-all inhibition (circles) decorrelates the neurons for learning and selects the strongest response for classification. Figure by Shih-Chii Liu, reproduced with permission of author.

Interfaces Interface boards are essential to route spikes from one module to another (mapping) as well as for testing and characterization (monitoring and sequencing). Based on the PCIAER board developed in Rome (called 'generation 0' in CAVIAR), a variety of interface boards has been implemented by the group of Anton Civit, Dep. of Computer Architecture and Technology, University of Sevilla (USE). In most cases the CAVIAR boards are portable, except the high-speed PCI-based monitoring and sequencing board (PCIAER generation 1). The portable boards are a first step towards autonomous AER systems. The USBAER board provides frame-based sequencing and monitoring, to bridge between a conventional representation and spiking systems. See Figure 4.5 for an overview and [Linares-Barranco et al., 2006, Berner, 2006] for details on the CAVIAR interface boards.

All modules in the CAVIAR project use the same AER protocol definition, that is an active-low point-to-point standard of 3.3V. In principle, the building blocks are interchangeable and can be assembled in different architectures. To focus the development, the CAVIAR system specifies a system architecture that serves as a demonstrator. In the next section we will describe the functionality of the system.

4.3 Experiments

We will first describe the initial interfacing of the individual building blocks, the object chip and the convolution chip, to show their functionality. We then present the complete processing chain in its first and second version.

4.3.1 Interfacing Object Chip to Convolution Chip

We interfaced the object chip to the convolution chip in an experiment with the IMSE partner in 2004. Figure 4.6 shows the test setup. We controlled the setup by two independent computers: One generates the test stimulus and configures the convolution chip (left side), the second displays the spike rates from the object chip and configures it (right side). The convolution chip is configured by specialized FPGA hardware, the object chip is configured by analog bias voltages that are generated with the dacboards and the ChipDatabase described in Section C. The second version of the object chip was used for this test. Figure 4.7 shows a picture of the setup.

We used two of the CAVIAR AER interfaces for the test: the PCIAER board in the 'random generator mode' in which it converts an input vector to Poisson spike trains, and the USBAER board in the framegrabber mode, that is the board generates a vector of spike counts.

To display the input to the object chip without reconnecting the setup, we disabled the inhibitory connectivity in the winner-take-all chip and recorded the output of the object chip. This shows the effective input, that is the synaptic

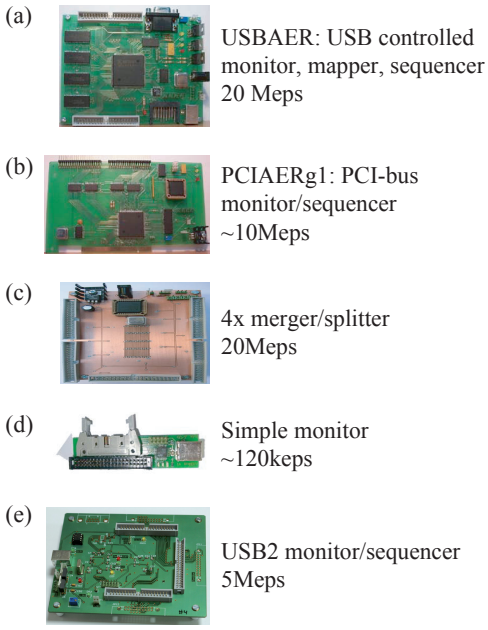

Figure 4.5: Interfaces developed in the CAVIAR project. (a) USBAER board, the most versatile portable board because it can be easily programmed as a mapper, monitor and sequencer. It provides frame-based and spike-based monitoring modes (framegrabber and datalogger). A variety of extensions has been developed, for example a probabilistic mapper, see Section 3.3.2. (b) The PCIAER board generation 1 provides high-speed spike-based monitoring and sequencing capabilities. (c) Merger/splitter board to combine multiple AER channels. Up to 4 inputs or outputs can be merged or split. Connectivity schemes include broadcast or unicast modes. In the latter, the uppermost two address bits specify the channel. (d) Simple monitor board, a bus-powered monitor board with a microcontroller that is capable of recording slow spike rates. (e) The USB2 board can monitor and sequence spike trains with frequencies of up to 5MHz to and from a computer using the USB2 protocol. Pictures (a) to (d) adapted from [Serrano-Gotarredona et al., 2005], (e) Picture by Tobi Delbruck, reproduced with permission of author.

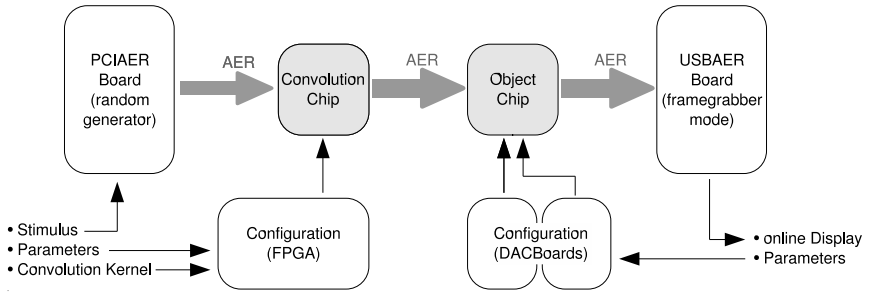

Figure 4.6: Overview of the test setup. The PCIAER generation 1 board is used to generate AER artificial stimuli which are transmitted to the convolution chip. The output of the convolution chip is directly connected to the object chip. Output spikes of the object chip are recorded as spike count frames with the USBAER board and displayed. Configuration of the convolution chip is done with a special configuration FPGA board, while the bias settings of the object chip are generated with dacboards.

Figure 4.7: Picture of the test setup. From left to right: convolution chip in the CAVIAR standard PCB, object chip with two dacboards attached, USBAER framegrabber board, laptop with on-line spike rate display.

input with mismatch in the synaptic efficacies. Compared to the output of the convolution chip, the spike rates are divided by the number of spikes n the neurons need to reach threshold.

The kernel of the convolution chip was programmed with a ring-shape pattern, see Figure 4.8. The stimulus was constructed to match the convolution kernel, by centering the ring structure at pixel position (4;7). The spike rates of the neurons active in the ring were set to their maximum spiking frequency.

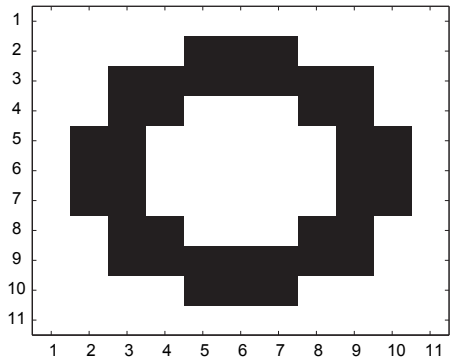

Figure 4.8: Ring-shape of size 11x11 used for testing. We used this pattern for both the convolution kernel and the stimulus. Convolution kernel: pixels shown in black represent maximal excitatory synaptic weights, white pixels represent zero weights. Stimulus: the ring is centered on pixel position (4;7), and the neurons shown in black were set to spike with maximum rate.

The convolution operation results in the a pyramid-like shape, see Figure 4.9a. The winner-take-all correctly selects the strongest input and suppresses all other outputs. Since we use the winner-take-all with all quadrants combined, both inhibitory neurons of all 4 subarrays show activity (Figure 4.9b). See Section 3.2 for a discussion of the combined winner-take-all mechanism.

For this experiment, the kernel contained only excitatory weights. If both excitatory and inhibitory weights are used, the kernel matches the stimulus exactly (matched-filter stimulus) and the output of the convolution chip can be adjusted to directly detect the center of the stimulus, with only minor outlying activity. On such clean input, the winner-take-all network removes the remaining outliers easily, see Figure 4.10.

4.3.2 Complete Processing Chain

At the CAVIAR project meetings in April 2005 and February 2006, the complete processing chain was assembled. The system consists of the retina, the convolution chips, the object chip and the learning module. To route the spikes from

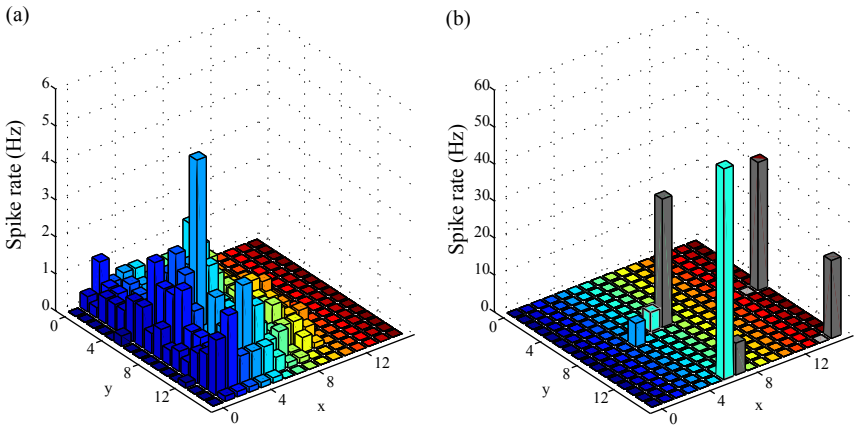

Figure 4.9: (a) Input to the object chip for a matched-filter response of the convolution chip. The convolution chip was programmed with the ring-shape stimulus as shown in Figure 4.8, while the stimulus contained the same shape centered on pixel position (4;7). The matched-filter response results in a pyramid-like output (we will later approximate a Gaussian shape) that is affected by the mismatch in the input synapses of the object chip. (b) Winner-take-all output, correctly selecting the strongest input at position (4;7) and suppressing all other output. The active neurons in the corners of the quadrants are the inhibitory neurons (light gray: first inhibitory neurons, dark gray: second inhibitory neurons). Spike rates are not measured accurately due to an overflow in the framegrabber counter, that is the spike rate is measured module the counter size of 255. From theory, the activity of the first inhibitory neuron (position 6;15) is equal to the activity in the array (position 4;7).

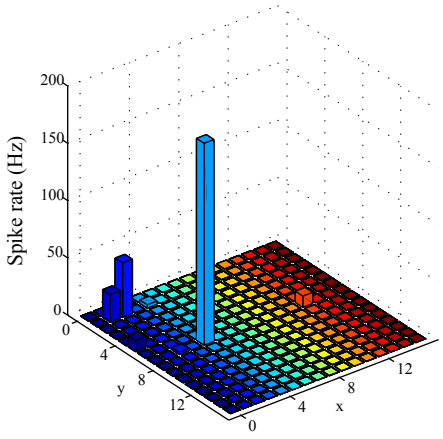

Figure 4.10: Convolution operation with inhibitory weights in the kernel. Including inhibitory weights in the kernel of the convolution chip results in a very clean output, showing the stimulus center and minor outlying activity at the boundaries of the integration kernel. Effective input to the winner-take-all network.

one module to another, mapper (USBAER boards) and merger/splitter boards are inserted. In the first version of the chain (see Figure 4.11), framegrabbers were used to display the rate activity of the neurons. On the second version of the chain we could record the spike trains including all timing information with the USB2 boards. Figures 4.12 and 4.13 show pictures of the test setup of both chains.

Figure 4.14 shows the output of the system. Input to the artificial retina comes from objects moving with constant speed (here black disks on a white background that rotate with constant speed). The retina codes the temporal contrast edges in a spiking representation (left inlay): white dots represent 'ON' events, that are responses to positive contrast edges, black dots represent 'OFF' events, that are in response to negative temporal contrast edges. The spikes are transmitted to a set of convolution chips. Here the spike-based convolution is programmed to detect the center of the disk that best matches the convolution kernel (middle inlay). Four convolution chips are tiled to increase the resolution. Positive hits of the convolution operation are represented as white dots, negative hits as black dots. The output of the spatial filtering is cleaned by the 'object' chip, which performs a winner-take-all which detects the current object position using the winner-take-all operation (right inlay). White dots mark the spiking output of excitatory neurons of the object chip; black dots show the activity of the inhibitory neurons involved in the computation. The position of the detected disk is expanded over time in the delay line chip, and the resulting

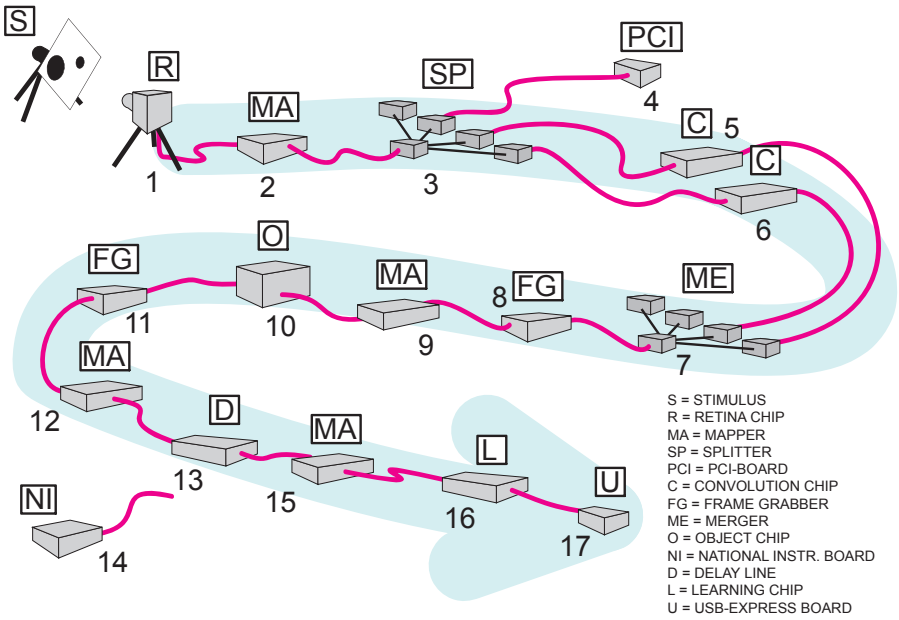

Figure 4.11: CAVIAR setup. Input to the artificial retina is a rotating white disc with black circles (S). The retina (R) produces spikes that code temporal contrast edges. These spikes are routed through a mapping table (MA) and a splitter board (SP) to a set of convolution chips (C). In the first version, two convolution chips were used; in the second version four. Output from the convolution chips is merged (ME) and remapped to the object chip (O). In the first version of the chain, a PCIAER board (PCI) monitored the spikes from the retina, while framegrabbers (FG) recorded the spike rates further down the chain. In the second version, USB2 boards were used for all measurements. Output of the object chip is remapped to the delay line chip (D). An additional signal card (NI) was used to configure the delay line chip. The expansion into time from the delay line is routed with another mapper to the learning chip (L). While on the first version a simple USB monitor (U) was used to record the output, in the second version the output controlled a spiking actuator. Figure by Patrick Lichtsteiner, reproduced with permission of author.

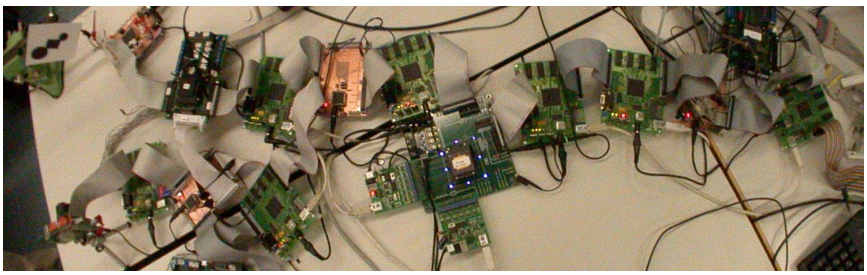

Figure 4.12: CAVIAR demonstrator, first version, setup picture. The modules are arranged as described in Figure 4.11. Shown is: stimulus (top left corner), retina (bottom left corner), one of the convolution chips (3rd from top left), object chip (middle, with blue LEDs and dacboards), delay line chip (top right corner). All other boards are USBAER interface boards and merger/splitters. Picture by Tobi Delbruck, reproduced with permission of author.

trajectories are learned by the learning chip. The output of the learning chip is one of its 32 neurons that span the presented input space.

CAVIAR makes use of the versatile coding of a spike: along the hierarchy of processing, a spike encodes temporal contrast change (retina), an output of the convolution (convolution chips), the estimated location of the detected object (object chip) and the detection of a learned trajectory (learning module). The occurrence of a spike encodes the time of the event, the address encodes the location, while the type of the event (for example, contrast change) is determined from the output of which stage the spike is recorded from.

The second version of the CAVIAR system was stable enough to make experiments with different stimuli possible. We initially started with black disks on a white background that rotates with constant speed, see Figure 4.15a. We varied the size and number of the disks as well as the shape of the 'objects' attached to the disk. In the next experiment we moved a disk in the direction of the z-axis, perpendicular to the viewing plane of the retina. The four convolution chips were programmed with kernels of disks in four different sizes. These feature maps were processed by the object chip. The group of winner-take-all networks first selects the strongest input within each of the four scales and then their winners compete on the level of the feature maps. Since the size of the stimulus does not change, the identity of the best matching feature map reveals the distance of the shown disk. To test the high-speed processing capabilities of CAVIAR, we used a stimulus that switches between two flickering LEDs. The switch could be used to measure the latencies in the system. We also explored real-world stimuli using oranges instead of rotating disks, see Figure 4.15b,c. An orange on a string as a pendulum served as a stimulus with a still predictable trajectory. For the most advanced stimulus, juggling with 3 oranges, a quantification of the trajectories is difficult. Still, the convolution output was shown to

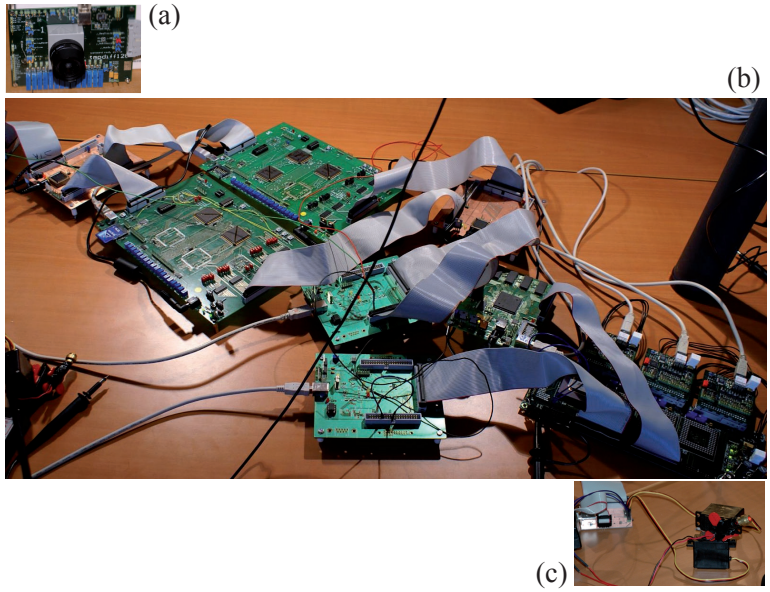

Figure 4.13: CAVIAR demonstrator, second version, setup picture. (a) 128x128 retina, (b) processing chain with set of convolution chips on two boards (2nd from top left) and object chip board (bottom right). Green boards are USB2 monitors; the others are a merger/splitter board and mapper. The learning module is not shown. (c) spike-based actuator with control board (left) and an actuator with two degrees of freedom with servo motors, carrying a laser pointer. Picture by Bernabe Linares-Barranco, reproduced with permission of author.

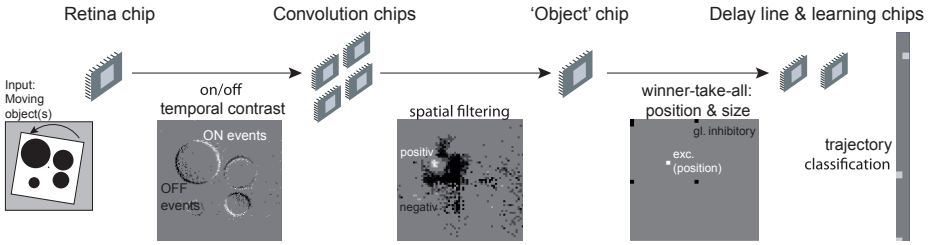

Figure 4.14: CAVIAR system output. (Top) hybrid VLSI chips performing the computation, (inlays) spike count representation of the output of the stages. Input is a rotating disk with black circles. Shown is the output of the retina, the convolution chips, the object chip and the learning module, which represents trajectories of the input object.

process the center of the oranges very clearly, filtering out the moving hands of the juggler.

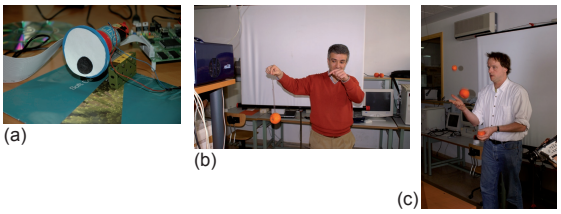

Figure 4.15: Stimuli used in CAVIAR. (a) rotating white background with black circles attached, (b) Bernabe Linares-Barranco holding a string with an orange attached to it into the field of view of the retina, in front of a white background, (c) Tobi Delbruck juggling with 3 oranges.

We used a simple actuator as shown in Figure 4.13c to close the sensory-motor loop for the first time in the CAVIAR system. The actuator was built by Rafael Paz Vicente. This simple system allows two degrees of freedom with two servo motors (yaw and tilt). It uses a microcontroller to translate the input AER address into position commands for motors. We used a resolution of 64 positions on each axis. This was the first time that we could close the sensory-motor loop in a completely spike-based system, that means all communication between sensor and actuator uses AER. We demonstrated that we could build a real-time system that draws the trajectory of a moving object onto a screen with a laser pointer.

4.3.3 Parameters on a System Level

The processing in CAVIAR depends on many parameters in the individual modules. In Figure 4.16 we list the main parameters that affect the object chip. The retina transforms the presented stimuli into spike trains. The functional parameters are the thresholds of the contrast changes that elicit spikes $\pm(\Delta I/I)_{\text{th}}$ and the refractory period t_{refract}, that is how fast the pixels can generate the next spike. The output of the convolution operation depends on the spatial properties of the programmed kernel, that is its size and shape, the effective synaptic weights w_{eff}[2] and the forgetting time constant t_{forget}. On the object chip, the winner-take-all operation depends mostly on the effective synaptic weight, that is the number of spikes n the neurons need to reach threshold. Further parameters are the strength of the self-excitation V_{self}, the leakage and any additional connectivity that incorporates context information. All these parameters modify the statistics of the spike trains and therefore the output of the object chip.

[2]With 'effective weights' we denote the change in the membrane potential normed to the membrane threshold, for example an effective weight of 0.1 means that the neurons needs 10 spikes to reach threshold.

As discussed in Section 2, the output of the winner-take-all can be characterized by the classification error e_{class}, the output rate ζ and the jitter in the output spikes e_{jitter}. These are the measures of the properties of the output of the object chip which influence the performance of the last stage, the learning module.

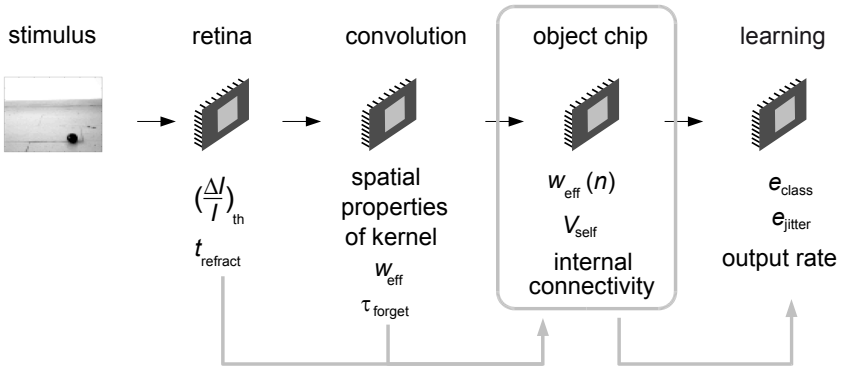

Figure 4.16: Parameters on system level of the CAVIAR architecture. We show the parameters of the stages that affect the computation in the winner-take-all, and how the performance of the winner-take-all affects the later learning module. In first approximation, the computation in the retina depends on the threshold of contrast change at which a spike is generated, and the refractory period, that is the recovery time a pixel needs to generate a subsequent spike. Computation of the convolution chip depends on the spatial properties of the programmed convolution kernel, the effective weights in the kernel and the time constant of the forgetting mechanism. The winner-take-all operation depends on the effective synaptic weights, that is the number of spikes a neuron needs to reach threshold. Further parameters are the strength of the self-excitation V_{self} and internal connectivity that incorporates context information. The performance of the winner-take-all, that is the predicted object location, can be characterized by the classification error, the jitter error and the output spike rate. We will discuss these measures in Section 4.4. This output of the winner-take-all is the base for computation in the learning module.

In the next section we will characterize the input and the output of the object chip. The input reflects the computation based on the parameters of the pre-processing stages, the retina and the convolution chip. The quality of the output reflects the performance of the winner-take-all chip in correctly predicting the position of the object seen by the retina.

4.4 Performance of Winner-take-all Stage

In this section we will compare the performance of the winner-take-all network in the CAVIAR system to the theoretical model we developed in section 2. We first discuss how well the input to the winner-take-all fits the assumptions we made in the theoretical model. Then we analyze the output of the winner-take-all. Having assured that the model is applicable to both input and output data, we can compare the performance predicted by the theory and the implementation in an aVLSI system that operates on real-world stimuli.

4.4.1 Input Statistics

In section 2.3.2 we developed a model of a winner-take-all network in which a wave of spikes is traveling along an one-dimensional chain of neurons. We assumed the waveform to be of Gaussian shape, and the spike trains within to follow the Poisson distribution. In this section, we will test these two assumptions on the CAVIAR data.

In the CAVIAR experiment that we consider here, the stimulus is a set of circles of different sizes on a white background that rotates with constant speed in front of the artificial retina. The convolution chip contains a matched-filter kernel of one size of a circle. Output spikes from the convolution chip indicate the detection of the convolution kernel. For this analysis we do not consider the off-spikes that indicate negative results of the convolution operation.

Taking mismatch and statistical variation into account, we can consider each output spike of the convolution stage as an event that denotes the presence of the stimulus in both space and time with a certain probability. Depending on the parameters of the convolution, the number of spikes and their relevance will change, resulting in a different shape of the average spike waveform that forms the input to the winner-take-all network.

In the analysis of section 2.3.2 we considered the neurons to be arranged in a one-dimensional chain. We transform the two-dimensional CAVIAR input into one dimension by considering only neurons along the trajectory of the stimulus (Figure 4.17). This discards activity from neurons outside the trajectory of the stimulus center. In this experiment, these outliers received less input than neurons on the trajectory and did not evoke output spikes of the winner-take-all. Our analysis focuses on the spatio-temporal estimation of the stimulus position, for which only the neurons with a significant spike input are relevant.

To select input along the trajectory of the stimulus, we manually defined a spatial region of interest consisting of a ring of one pixel width. Intersection of this ring with the spatial representation of the input spike train selects only input along the trajectory (Figure 4.17, right).

To assess how well the input data to the winner-take-all fits to our theoretical model, we examine the average input to each neuron. Since the neurons are aligned to a certain stimulus position, the input to each neuron corresponds to

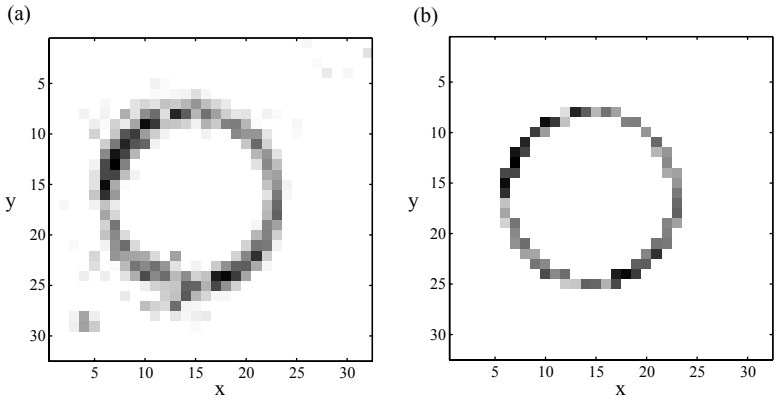

Figure 4.17: Spatial trajectory of the stimulus center. The stimulus is a disc that rotates with constant velocity in front of the artificial retina. The convolution stage contains a matched-filter kernel; its output is a smoothed version of the center of the object (a). The gray level gives the spike count for one revolution of the stimulus. For the analysis, we consider only pixels that fall onto the trajectory of the stimulus (b). We masked these pixels with a manually defined region of interest.

a movement step of the stimulus. We can describe the input in response to this step by the Peri-Stimulus-Time-Histogram (PSTH), a commonly used method to describe spike train responses. In the PSTH, spike trains from different trials are aligned to the stimulus onset. The histogram is obtained by binning all spikes with a given bin time. The resulting histogram is normalized to the number of trials, resulting in the average neuronal response to the stimulus. In our case the stimulus is rotation symmetric, so every each input channel can be seen as one trial of the same experiment. The PSTH is obtained by averaging over all input channels and multiple revolutions of the stimulus.

Reconstructing the waveform from one only input is difficult, as is illustrated by an example in Figure 4.18. We will use the PSTH to average over all input channels.

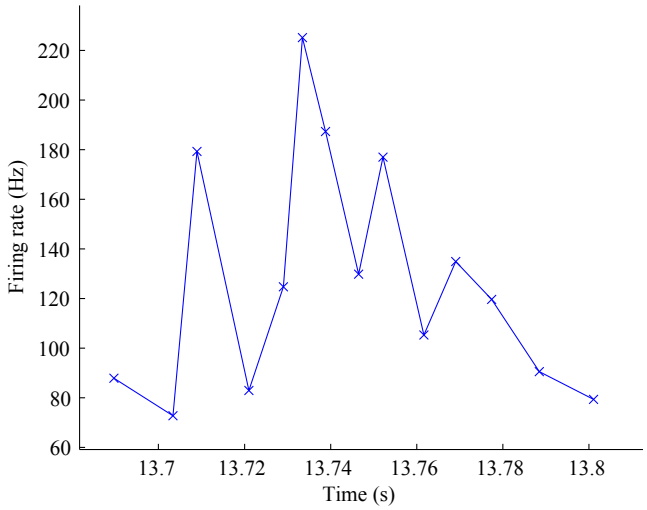

Figure 4.18: Estimation of waveform of the convolution output, using only one spike train. For each inter-spike-interval $\text{isi}_i = t_{i+1} - t_i$ between two spikes at t_i and t_{i+1}, the firing rate is calculated $(1/\text{isi}_i)$ and plotted at t_{i+1}. Due to the variation in the times of the spikes, the resulting waveform is quite noisy.

In the CAVIAR experiment the position of the stimulus was not recorded, but has to be estimated from the data. We explored the following methods:

Linear movement. By assuming the stimulus moves at a constant angular velocity, the time of the alignment of the stimulus to the input channels can be calculated. The center of the rotation is estimated from the center of the spatial map (Figure 4.17). The angular velocity was estimated from the input spike trains. For each input channel, the estimated time of alignment with the stimulus is calculated. We aligned all input channels to the estimated time of alignment. The resulting alignment was not good enough to perform a PSTH

estimation. This is most probably because the angular velocity of the stimulus is not constant, which is reasonable if we consider the simple stimulation method with a non-calibrated motor. We will therefore estimate the stimulus alignment directly from the data.

Median spike alignment. Aligning the input channels by their median removes variation of the stimulus velocity to some extent, but suffers from another source of error: for the following binning of the PSTH, every spike train has a spike in the center bin, that is at this time all spike trains are correlated. According to Poisson generation of the spikes not every spike train would have exactly one spike at the peak of the rate function. We therefore use a method that does not center the input spike trains at a common time.

Mean spike time alignment. This is achieved by aligning the spike trains by the mean of their spike times. With the variation of the spike trains, the mean spike times will be shuffled so that no correlation is introduced in the spike trains.

The alignment described above can only work if the spike trains do not contain too many outliers, that is spikes that are far apart from the overlap time with the stimulus. In the experimental data used here there are only single outlying spikes. We cleaned the spike trains from these outliers by iterating the alignment procedure: first, all spike trains are aligned by their mean. We then discard the outliers by deleting all spikes with timestamps more than 200ms apart from the mean. The first alignment is necessary to estimate the width of the stimulus waveform and to determine the cut-off boundary. In the second step, the spike trains with the remaining spikes are realigned, resulting in a better alignment with outliers removed.

Figure 4.19 shows the spike trains before alignment, but sorted by their mean spike time. From this representation the average traveling time d from one neuron to the next can be calculated, by averaging the difference the mean time between each pair of neighboring input channels (see Table 4.2). Figure 4.20 shows the spike trains after alignment.

From the aligned data set the PSTH can be calculated by binning and normalizing to the number of neurons and the bin time. Before we do this, we have to check that all spike trains have about the same length, otherwise the PSTH would be falsified (Figure 4.21).

The resulting PSTH is shown in Figure 4.22, together with a Gaussian fit. The goodness of the fit is quite high. From the fit, the parameters σ (standard deviation) and peak spike rate r_{max} are extracted. To quantify the goodness of the fit, we extracted the parameters separately for each input channel and then averaged over the results of each spike train, see Table 4.2.

To determine the variation in the spike times, we have to transform the non-stationary spike trains into stationary ones. We use the time-rescaling theorem [Brown et al., 2002, Kass et al., 2005]:

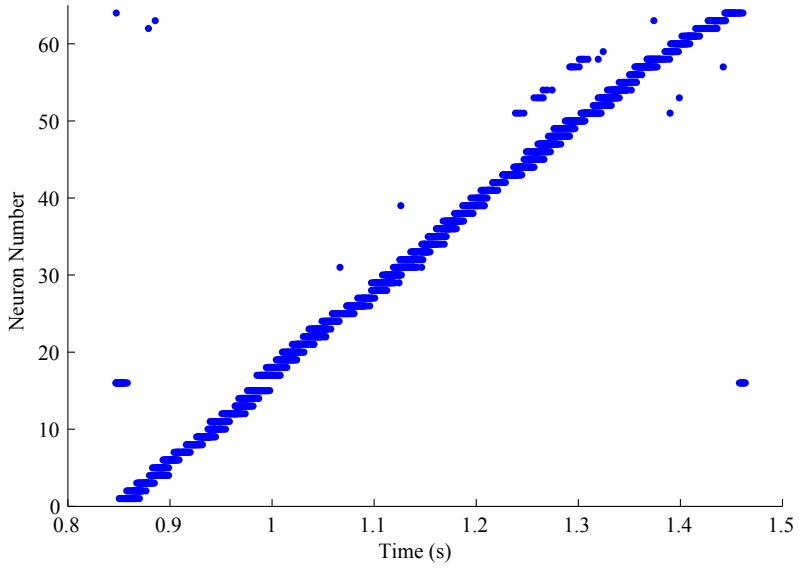

Figure 4.19: Raster plot of the input spike trains. Input channels along the trajectory are sorted in the order of the stimulus movement. Each point marks one spike (every spike train contains about 20-40 spikes). Data from one revolution of the stimulus disc are shown.

Parameter	CAVIAR	simulated Poisson
d	95ms ± 31%	95ms ± 9.5%
σ	46ms ± 11%	46ms ± 4.5%
ν_{max}	373Hz ± 37%	372Hz ± 17.6%
CV	0.83 ± 23%	0.96 ± 16%

Table 4.2: Parameters of sample CAVIAR data set for Gaussian-shape traveling wave. The percentage values give one standard deviation. For comparison, the parameters of an artificially created traveling wave of Poisson spike trains with the same means are given on the right side.

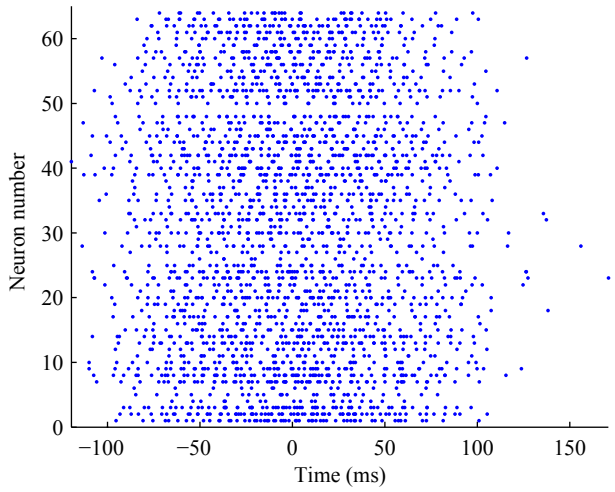

Figure 4.20: Raster plot of the input spike trains after alignment. Input channels are aligned to their mean, as described in the text. Each point marks one spike (every spike train contains about 20-40 spikes). Data from one revolution of the stimulus disc are shown.

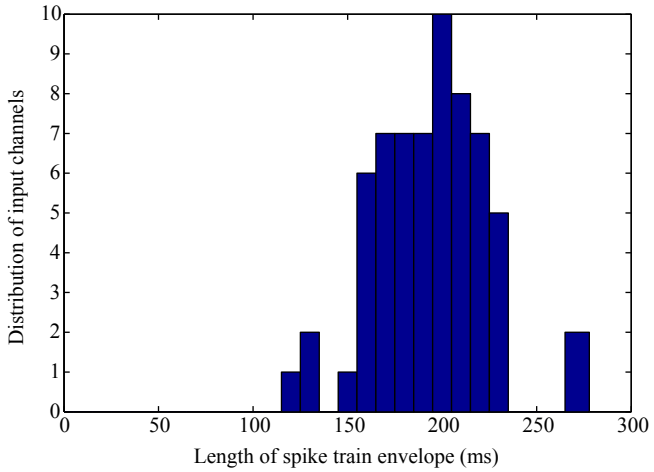

Figure 4.21: Histogram of the lengths of the spike train envelope of each input channel. From the cleaned input spike trains, minimum and maximum spike times are extracted to compute the length of the envelope. The envelope lengths are distributed about the same mean; otherwise the PSTH would be corrupted. Data from one stimulus revolution.

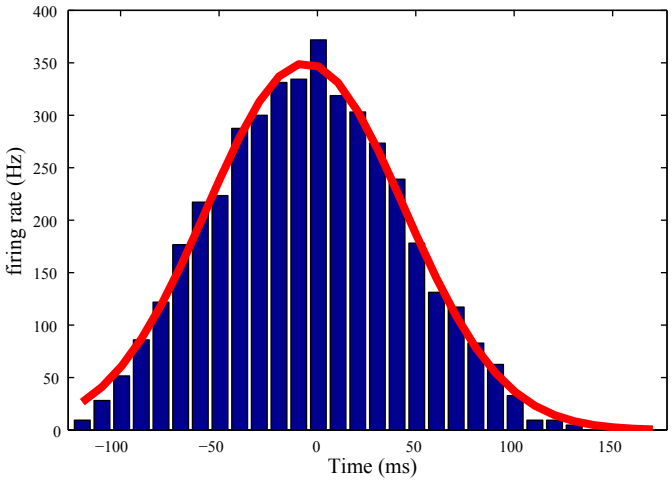

Figure 4.22: Peri-Stimulus Time Histogram (PSTH), averaging over about 2500 trials (60 neurons and multiple revolutions of the stimulus). The red line shows a Gaussian fit.

Time-Rescaling Theorem. *Let* $0 < t_1 < t_2 < \ldots < t_n < T$ *be a realization from a point process with conditional rate function* $\nu(t)$ *satisfying* $0 < r(t)$ *for all* $t \in]0, T]$. *Define the transformation*

$$\Lambda(t_k) = \int_0^{t_k} \nu(u)\mathrm{d}u, \tag{4.1}$$

for $k = 1 \ldots n$, *and assume* $\Lambda(t) < \infty$ *with probability one for all* $t \in]0, T]$. *Then the* $\Lambda(t_k)$*'s are a Poisson process with unit rate.*

For proof, see reference [Brown et al., 2002].

In our case we can generate new spike times $\Lambda(t_k)$ if we know the rate function $\nu(t)$ that generated the original spike times t_k with a point process. The Poisson process is an unconditional point process as required by the theorem. As rate function we use the Gaussian fit to the PSTH as obtained from the averaged trials. It fulfills the condition $0 < \nu(t)$ for all $t \in]0, T]$ and is a smooth function as discussed in [Kass et al., 2005]. The parameters of the Gaussian function are the mean spike time μ and the standard deviation σ. We defined the Gaussian function in Equation 2.43. The integration limits are the times of the first and last spike.

$$\hat{t}_k = \int_{t_0}^{t_n} \nu_{\mu,\sigma}(t)\mathrm{d}t \tag{4.2}$$

The transformed spike times $\hat{t}_k$ form a homogeneous spike train that allows us to test for Poisson statistics. We calculate the coefficient of variation (CV) as the standard deviation of the ISIs by their mean, see table 4.2. A coefficient of variation close to one is a necessary condition for a Poisson spike train, but not sufficient. If the spike trains follow a Poisson distribution, the inter-spike intervals should follow an exponential distribution. Figure 4.23 shows the distribution of the inter-spike intervals. To understand the distribution shown in Figure 4.23, we have to discuss what kind of errors are introduced by the processing the CAVIAR chain.

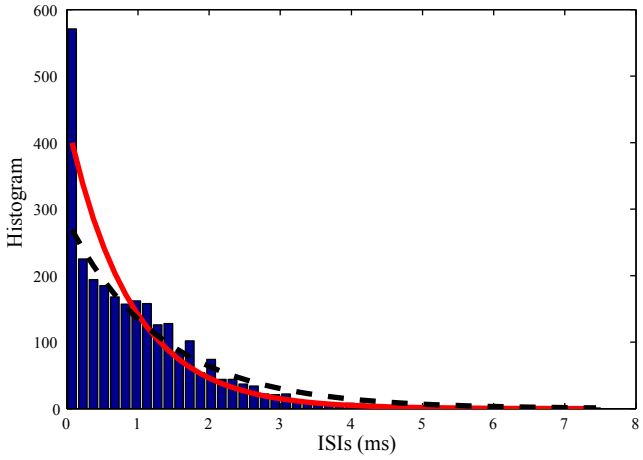

Figure 4.23: Inter-spike interval distribution after transforming the spike trains into a homogeneous process. Data from one stimulus revolution. The continuous lines show exponential fits to the complete data set (continuous line) and excluding the minimum inter-spike interval (dashed line).

Spike trains in the CAVIAR system exhibits two types of variation: the first type changes the spike rate or the number of spikes that are generated, for example for a contrast change in the retina. The second changes the latency of the spikes, resulting in spike jitter. We assume these errors to follow a normal distribution, or a skewed normal distribution as we analyzed for the mismatch intrinsic in analog VLSI, see Section 3.3. In Section 4.3.3 we discussed the system parameters that affect the computation before the winner-take-all. All parameters except the spatial properties of the convolution chip are subject to mismatch induced by the analog VLSI implementation. In addition, the angular velocity of the stimulus varies slightly since we used an uncalibrated DC motor. Differences in the illumination lead to varying contrast changes that are recorded by the retina.

It is not clear to us why these variations would add up to a Poisson distri-

bution. The law of rare events states that the Bernoulli distribution converges to a Poisson distribution for a large number of events and a small probability of occurrence. Still, our data shows that the distribution of inter-spike-intervals can be approximated by an exponential function (see Figure 4.23, dashed line), which implies a Poisson distribution of the output spike trains of the convolution chip.

On top of this distribution, sub-sampling is another source of variation: from the retina resolution of 128x128 the input channels are sub-sampled to 64x64 for the convolution stage, and 32x32 for the winner-take-all stage. Sub-sampling combines neighboring input channels. Since these originate from spatially related stimulus input, neighboring spike trains exhibit correlations, that is spikes happen closely after each other. When merged during sub-sampling, this results in very small inter-spike intervals which are reflected in the peak at minimum ISIs in Figure 4.23.

A third influence on the statistics of the input spike trains is the integration in the neurons of the convolution stage. Since several spikes are integrated before the threshold is reached, a Poisson distribution would be transformed in to a more regular ISI distribution.

Although the input spike trains do not accurately follow a Poisson distribution, the Poisson approximation still fits our data set quite well, at least with an over-imposed peak at minimum inter-spike-intervals induced by the sub-sampling. Furthermore, the Poisson distribution is one of the most commonly used distributions to model spike train statistics. We continue our analysis of the CAVIAR data with keeping in mind that the true inter-spike interval distribution deviates from Poisson statistics to a certain degree.

4.4.2 Analysis of Output

Figure 4.24 shows the output of the winner-take-all network in the CAVIAR experiment for which we analyzed the input data. Two variations can be seen: (1) variation in the number of output spikes per position, and (2) variation in the angular velocity of the stimulus disc.

(1) As shown in the inlay of Figure 4.24, the number of output spikes per position varies strongly. The large variation is caused by the mismatch in the synaptic efficacies (about 20% on this third chip version), and the variation in the input spike rates. The input spike trains show significantly more variation in the peak spike rates than expected for a Poisson distribution, as we have shown with an artificially created spike train in Table 4.2. This variation in the input is reflected in the output of the winner-take-all. The standard deviation of the peak rate is 37% and the variation in the width of the spike wave 11%. The three independent error sources, including the mismatch in the synaptic efficiacies, add up to 68%. In the experiment, the winner-take-all network was tuned to generate at least one spike per stimulus position. On average each neuron generates 5.3 output spikes with a standard deviation 3.8 spikes, that is

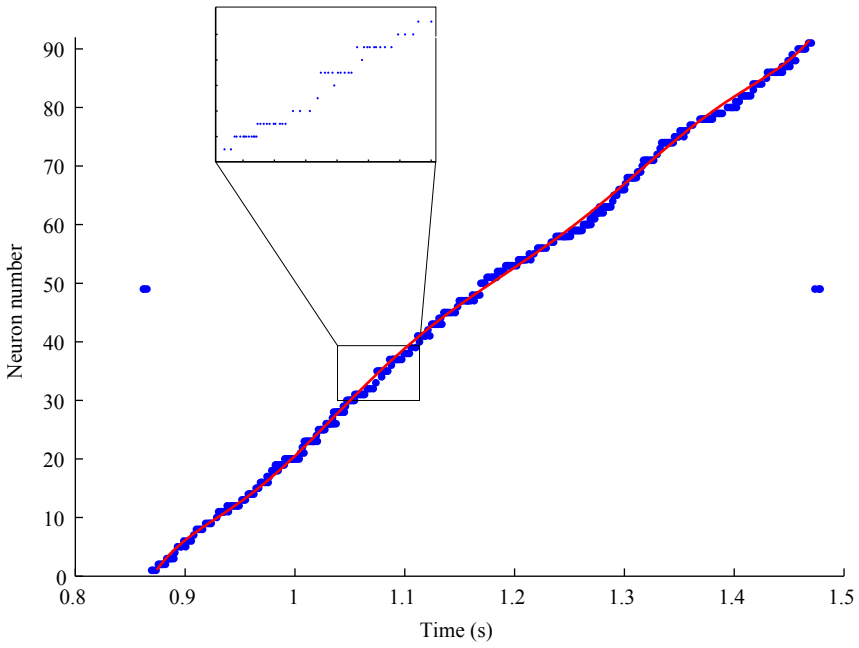

Figure 4.24: Output of the winner-take-all network in the CAVIAR experiment. Neurons are sorted in the order of the movement direction of the stimulus. Each dot marks one spike. Every spike train contains about 2-9 spikes. Data during one revolution of the stimulus disc is shown. Neuron 49 has outlier spikes which are not significant for the later analysis. The number of output spikes per position varies strongly,as can be seen in the magnification of the inlay.

a coefficient of variation of 72%. The variation in the output data is therefore only slightly higher than the expected sum of the variation in the input.

(2) The variation in the angular velocity of the rotating stimulus disc is now visible. In the input data the variation was hidden by the large number of spikes that were produced at each position, see Figure 4.19. We approximated the stimulus position by smoothing a polynomial function to the data (Figure 4.24, red line). As we will discuss, this approximation is a critical step in the estimation of the winner-take-all performance.

Variation induced by the Poisson distribution of the input results in small-scale errors that do not introduce a systematic mismatch to the stimulus position (it does not introduce an offset, for example). We can therefore reconstruct the stimulus position from the data without being affected by the Poisson variation if we use a smoothing function that only considers the large-scale structure of the stimulus. We tested polynomial smoothing functions of degrees 5 to 9. For a polynomial fit with a degree of less than five, the large-scale variation of the stimulus position is not captured good enough; for degrees higher than nine also the small-scale variation of the Poisson variation is smoothed.

To quantify the performance of the network we use the area error as discussed in Section 2.3.2. We defined the area error e as the area between the predicted ball position, a staircase function since the position is discretized to the neurons of the network, and the position as indicated by the output spikes of the winner-take-all. Figure 4.25 illustrates the procedure.

We calculated the area error e for different smoothing functions of the ball positions, that is for smoothing with polynomials of degree five to nine. The resulting area errors are listed in Table 4.3. The choice of the smoothing function affects the area error significantly, with a difference of 25% between finest (0.60) and coarsest (0.75) smoothing.

To compare the performance of the chip to our model, we calculate the theoretical performance for our model developed in Section 2.3.2 with the parameters extracted from the input data as shown in Table 4.2. From the peak spike rate $\nu_{\max}$ and the width of the spike wave σ we derive the optimal number of spikes the neurons need to reach threshold $n = 22$ (Equation 2.45). The classification performance, that is the probability of a correct object location, is given as $P_{n,d,\sigma} = 0.85$ (Equation 2.46). The jitter error, that is the deviation of the output spikes from the time of the alignment of object and neurons, is $e_{\text{jitter}} = 0.33$ (Equation 2.49). The total area error combining classification performance and jitter error is $e = 0.64$ (Equation 2.51). All values are summarized in Table 4.3.

The implementation in the CAVIAR system reaches about the same performance as the prediction by our model. If a 7^{th} order polynomial is used to extract the object position from the data, the area error in the CAVIAR experiment and the model is equal ($e = 0.64$).

However, we do not think that a comparison in absolute numbers between our model and the CAVIAR experiment makes sense. There are too many as-

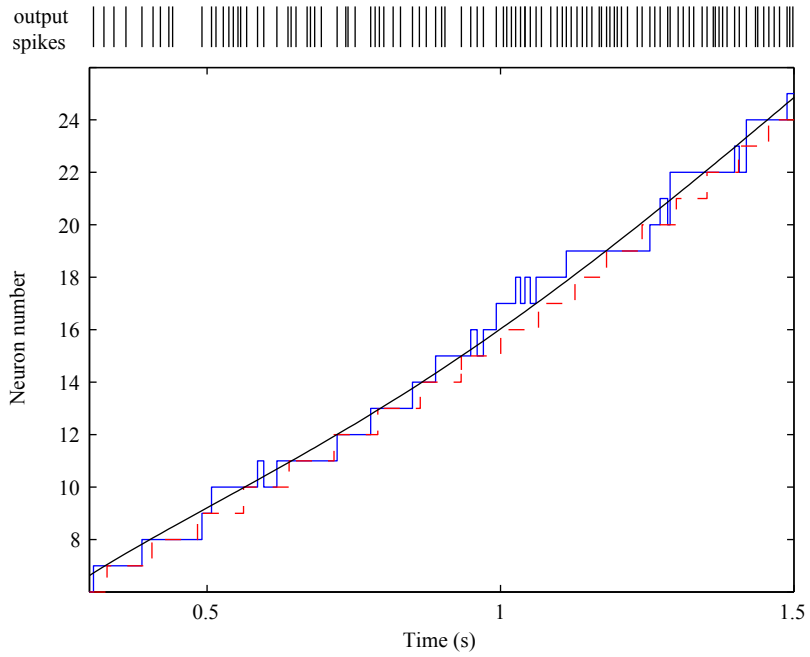

Figure 4.25: Reconstruction of object position from the winner-take-all output spikes, CAVIAR data. The position of the stimulus is reconstructed from the data (continuous line), see text. The ideal output of the network is an update of the object position as soon as the object is aligned to a new neuron (dashed staircase). In the most sparse representation the network would elicit one spike at each vertical line of the dashed staircase function. In the CAVIAR data (continuous staircase), the winner-take-all network elicits more spikes than one spike per position, as illustrated by the output spike train of the network at the top. This leads to switching in the predicted object position, for example between neurons 17 and 18. In addition, the output sometimes indicates an incorrect position, for example at neurons 18 and 19, or the spike times are jittered, for example at neurons 13 and 14. The area error e measures these errors by considering the area between the predicted ball position (dashed staircase) and the actual prediction from the data (continuous staircase), normalized to one neuron.

Performance from data:		
total area error e	'poly5'	0.75
	'poly6'	0.71
	'poly7'	0.64
	'poly8'	0.61
	'poly9'	0.60

Performance from model:	
n	22
classification performance $P_{n,d,\sigma}$	85%
jitter error e_{jitter}	0.33
total area error e	0.64

Table 4.3: Comparison of the winner-take-all performance for the data from CAVIAR experiment and the model developed in Section 2.3.2. The total area error e is the area between the actual object position and the position as indicated by the output spikes of the winner-take-all. For the experimental data, e varies with the degree of the smoothing function used to extract the object position from the data, see text for discussion. Still, the area error from the data is in the same range as the prediction by the model.

sumptions that falsify the accuracy of both model and experimental data. For our model this is the assumption that integration in the neurons always starts at the optimal time point, that is the intersection of the input waveforms to neighboring neurons. Second, the definition of the classification and the jitter error as an area counts some areas twice as discussed in Section 2.3.2. These problems can already be seen in the comparison of model and simulation results in Figure 2.15.

Calculating the area error from the CAVIAR data also has difficult assumptions. First, we regard a two-dimensional retinotopic stimulus as one-dimensional by using only the neurons along the trajectory. We do not consider the distortion that is introduced by this transformation. Second, our data cleaning removes spikes originating from neurons outside the trajectory which might have influenced the winner-take-all computation. The most severe problem is the extraction of the object position from the same data that we use to calculate the accuracy in the output of the position, by using a approximation function that smooths the small-scale variation the we are interested in. In addition, assuming the input to be Poisson distributed is only an approximation as we discussed before. Taking these different error sources into account, the close agreement of performance of the winner-take-all in the CAVIAR data and in the model is quite good.

We regard the comparison of model and data as qualitative. Effort could be undertaken to increase the accuracy of both model and data analysis. For ex-

ample, different starting times of the integration could be incorporated into the model. The experiment could consider data that did not involve sub-sampling. However, the model was originally designed to give an intuitive feeling about the principle of winner-take-all operation; and any further detail would complicate the formulas used. The analysis of CAVIAR data aims at a different purpose, that is a quantitative measure of the performance of the circuit to tune the complete system for optimal performance. It allows us to quantify different parameter settings in the winner-take-all network, and different parameters for the processing along the CAVIAR chain, for example for retina and the convolution chip. The effect of these modifications can be evaluated by the performance of the last stage before learning is involved, that is the performance of the winner-take-all to measure the correct object position.

In this chapter we presented a method to quantify the performance of the implementation of the winner-take-all according to the theoretical model we developed. Emphasis was put on the evaluation how good the input data fits to our assumption of a Poisson traveling wave. The CAVIAR data shows more variation in the waveform than an artificially created spike waveform. We quantified the Gaussian wave with the mean distance between the subsequent position, the width of the waveform, and the peak spike rate. Taken the variation in the stimulus, the sensor and the computation of the convolution into account, the input data to the winner-take-all can be approximated quite well by a traveling Gaussian waveform. The CAVIAR spike trains follow a Poisson distribution overlaid by correlations that are induced by the subsampling in the preprocessing. With this characterization of the input statistics we were able to compare the output of our theoretical model to the implementation. Although the area measure as the difference between estimated and measured position does not capture all properties of the output spike train, it allows a quantification in the stimulus position as predicted by the network. The CAVIAR system comes close to the limit in the performance that is induced by the Poisson nature of the input spike trains.

Optimal performance of the winner-take-all is obtained if the number of spikes the neurons need to reach threshold matches the stimulus properties, as stated by the theory. For the Gaussian traveling spike wave, this depends on the peak spike rate and the width of the profile. In the CAVIAR system, these properties change if the convolution kernel changes. In principle, the statistics of the retina output are preserved or change only slightly if the illumination or the speed of the stimulus changes, since the retina output encodes local contrast changes. However, in our experiments the spike statistics change if speed and trajectory of the stimulus are varied significantly. To obtain optimal peformance, the winner-take-all properties have to be adjusted, that means the synaptic weight of the neurons is retuned to match the width of the Gaussian profile.

Chapter 5

Conclusions

In this work we explored a spike-based neuronal network in the aspects of theory, implementation and application. The chosen winner-take-all network, is an ideal testbed for discussing computation in a system whose basic building blocks perform asynchronous event-based and time-continuous processing.

Asynchronous and time-continuous computation is one of the outstanding features of biological information processing systems compared to conventional computer architectures. As we discussed in the introduction (Section 1), we think that incorporating time into the basic building block of computation is an important key for building systems that are capable of interacting with the world in real-time. Taking inspiration from biological spiking neuronal networks to construct artificial systems is a logical step.

Applying the principles of biological processing to engineering applications requires a thorough understanding of the underlying computation, that is the processing architecture, the range of network and circuit parameters and the resulting performance. We approach such an understanding for the spike-based winner-take-all network.

Previous analyses of winner-take-all behavior have considered analog or spike-rate coded inputs (see Section 1.1 for an overview). These coding schemes are useful for stationary inputs, but fail to describe the dynamic input signals that occur in real-time spike-based processing systems for stimuli that vary with time. We are interested in the principles of winner-take-all computation in the transition between single spike and spike rate coding.

In the theoretical part of this thesis (Chapter 2), we derive mathematical formulations of the winner-take-all behavior. We first simplify the network to a minimalistic version in which two neurons act as spike counters and reset each other through strong inhibition. Starting from [Jin and Seung, 2002], who analyzed a winner-take-all network in the case of constant current inputs, we derive equations for the boundary conditions of the winner-take-all behavior in the case of input spike trains of regular frequencies (Section 2.1). For these

inputs, the network can select the winner with only one inter-spike interval, making the selection process very fast.

In biological systems, spike trains are often assumed to follow Poisson statistics. In that case, our equations determine the probability of the winner-take-all to correctly select the neuron that gets the strongest input. The performance depends on the number of spikes the neurons need to reach threshold, that is the time until the network reaches a decision, and the difference in the input spike rates. We first discuss the case of stationary input (Section 2.2), before we extend our analysis to time-varying input in which the stronger input signal switches from one neuron to the other (Section 2.3.1), and finally to dynamic input in form of a spike wave of Gaussian shape that travels across the neurons (Section 2.3.2).

We show that the decision of the network is optimal, that is it makes use of all the information available in the input spikes. Using information theory (Section 2.2.4), we discuss the winner-take-all network as a classifier that reaches a decision based on the information transmitted in the input spikes, and indicates its decision with output spikes. Depending on the number the neurons need to reach threshold, the winner-take-all increases the mutual information that is conveyed per spike about the stimulus, thus compressing the spiking representation.

We use our network model to examine the effect of properties typically found in biological systems. We quantify the network performance for weak inhibition, self-excitation, dynamic synapses and variation in the synaptic parameters. Weak inhibition decreases the network performance (Section 2.2.2), while self-excitation increases the performance for stationary inputs (Section 2.2.1). Dynamic synapses show only a slight improvement for the Gaussian inputs we consider (Section 2.4). Variation in the synaptic efficacies limits the discrimination performance, depending on the standard deviation of the synaptic parameters (Section 3.3.3). In the case of weak inhibition and dynamic synapses, the equations to describe the network performance do not hold anymore. Instead we use simulations of spiking neurons. Our network model and the discussed input cases set up the framework for exploring other neuron and synapse models in the future, for example the effect of conductance-based synapses.

What are the insights from our model for the processing in biological systems? The cortical microcircuit [Douglas et al., 1989] includes a soft winner-take-all operation where neurons cooperate to enhance consistent interpretations of the input, and compete to suppress inconsistent ones. The important properties of this soft winner-take-all computation are linear amplification, gain modulation, sustained activity and signal restoration as discussed in the introduction (Section 1.1). The hard selection of our winner-take-all model appears contrary to these soft properties on the first glance. But assuming biologically relevant Poisson variation in the input, our model of a hard winner-take-all implements 'soft selection', that is it reaches a fuzzy decision just due to the statistical properties of the input.

Linear amplification is an intrinsic property of our network if strong internal synaptic connections are used which result in strong inhibition. Self-excitation leads to hysteresis in the network, that means the output activity persists at the location of the recent strongest input activity (Section 2.3.1).

One feature of the microcircuit is to connect local decisions with other intra-area and inter-area sources in both a competitive and cooperative way [Douglas et al., 1989]. We discuss models for both cases: non-local cooperation is implemented with permitted sets, in which additional connectivity patterns incorporate context information. If excitatory connections are used, this can implement signal restoration, see Section 2.5.1 for a discussion. Non-local competition is implemented by the mechanism of competition across winner-take-alls, in which the individual winner-take-all networks represent feature maps that compete on a second level (Section 2.5.2).

The amount of detail at which biological neurons and synapses have to be modeled to achieve accurate results is a topic of discussion. We started with the reduction of the network and the neurons to a simplified form, in which the neurons count spikes and the inhibitory connections are strong. The performance of the network can be quantified with a single equation that is numerically solvable. We can then examine the effects of properties typically found in biological systems, for example weak inhibition and self-excitation, by comparing the resulting network performance. This quantification allows us to assess the importance of biological details to the accuracy of a model.

With this good understanding of the computational principles in the spike-based winner-take-all, we can implement it using Very-Large-Scale-Integration (VLSI) technology (Chapter 3). Neurons and synapses are implemented using analog transistor circuits, while spikes are represented as digital address-events (AER), see Section 3.1 for an introduction into analog VLSI and AER systems. We describe the details of the three chip revisions that we implemented in Section 3.2, and demonstrate that the performance follows the prediction of the theory for inputs of constant currents, regular rates and Poisson spike trains in Section 3.2.1.

We identified three challenges for current VLSI implementations: mismatch limiting the functional properties of the circuits and its compensation; difficult usage of VLSI systems due to the lack of common configuration interfaces; and missing programming methods to add adaptation and learning capabilities.

We addressed the mismatch in Section 3.3 by characterizing variation in the input synapses and its effect on the neuron population (Section 3.3.1). We discussed different schemes to compensate for the mismatch and to set the synaptic efficacies in Section 3.3.2. Using D/A converters to program the synaptic weight is the most versatile method and was implemented on the third chip version. In the winner-take-all network, mismatch limits the discrimination performance. With compensation using burst-length adaptation, the first chip version can detect a different in the input rates of 10% instead of 277% in the uncalibrated

case (Section 3.3.3).

To improve the usability, we developed the 'ChipDatabase' software, a tool to create a graphical user interface for controlling the bias voltages of the analog VLSI system based on the chip specifications, documented in Appendix C.

To configure and program spike-based neuronal networks we developed a hybrid software/hardware framework that is used to specify the connectivity, to analyze the spiking activity and to insert artificial spike trains into the system (Section 3.4). It allows the embedding of software agents in the spiking communication which perform learning and adaption functions. Both the Chipdatabase software packages and the spiking framework are now in regular use by the members of the Institute of Neuroinformatics.

With the address-event representation of spikes, neuromorphic systems combine analog computation with digital communication, forming hybrid analog/digital systems. With our framework that embeds software agents, we propose to further develop these systems into software/hardware hybrids to increase the complexity of the systems to be able to cope with real-world applications.

Finally, in Chapter 4, we analyze our hardware implementation of the winner-take-all network in a large-scale multi-chip vision application, the CAVIAR system.

The CAVIAR project is the largest multi-chip VLSI spiking system assembled until now with a total of 37920 neurons and pixels, combined in up to 17 separate boards. Each of the building blocks can process spike rates of up to 10Mspikes/s, but typical spike rates in the system are much lower as the information per spike is increased along the chain of computation. The system consists of an artificial retina, a bank of spike-based convolution filters, the winner-take-all network and a learning module that classifies trajectories of moving objects.

This complexity is reached by separating the different functions of the system into individual modules that communicate in a common infrastructure, since each building block is characterized and optimized on its own (see Section 1.2 for a discussion). The CAVIAR architecture follows both a conventional computer vision architecture and a design inspired by biology. Extracting features with convolution kernels and classification is a classical approach in computer vision. At the same time, the convolution and winner-take-all networks can be seen as a pair of simple and complex cell layers as we discussed in the HMAX network (Section 4.1). Furthermore, the communication and processing in the system is completely spike-based.

We first describe the building blocks of the system (Section 4.2), before we assemble them to the complete system (Section 4.3). We then analyze the performance of the winner-take-all in the system. We show that the output of the higher stages of CAVIAR can well be approximated with Poisson statistics, although retina and convolution filter are completely deterministic and exhibit only a small amount of variation (Section 4.4.1). When the system sees a moving object, the input to the winner-take-all network is a spike wave of Gaussian

shape that travels across the neurons. This is the same input that we used for our analysis of winner-take-all behavior in Section 2.3.2, and we can compare the predicted performance of our theoretical model with the performance of the implementation in a large-scale system. The achieved performance, that is the detection of the position of a moving object the system is stimulated with, is close to optimal in the case of Poisson statistics (Section 4.4.2).

One of the prospective application of the CAVIAR project is high-speed vision. Fast detection tasks do not allow the time to encode the inputs as spike rates, but have to consider the effects of single spikes. [Thorpe et al., 2004] describe a model that encodes and processes visual input with only one spike per neuron using a spike-order code. In our approach we analyze the transition from single spike codes to spike rate coding, by quantifying the performance of the classifier dependent on the number the input spikes the network needs to reach a decision, that is to elicit an output spike.

In its current version, CAVIAR does not incorporate learning or adaptation before the very last stage. Convolution kernels and parameters of the computation of retina, winner-take-all and the delay line are chosen manually. Although designed for a different hardware, the software framework we describe in section 3.4 can be used to extend the static CAVIAR framework. Simple algorithms can also be implemented inside the remapping infrastructure. Such an infrastructure has been demonstrated by the integrate-and-fire array transceiver system [Mallik et al., 2005] that provides dynamic routing and changes parameters of the computation based on simple learning rules.

The CAVIAR system provides a good test bed for analyzing data from a large-scale spiking system. Because the output data from this artificial systems with deterministic building blocks show the large variation of Poisson distribution, we can use the same model of winner-take-all performance to compare results in theory, in simulation and in the implementation in a real-world system.

With the simple actuator experiment described in Section 4.3.2 we showed that CAVIAR successfully closes the sensory-motor loop in a system whose processing is completely spike-based. Such a system offers the possibility of solving simple real-world tasks based on visual processing without the overhead of conventional frame-based architectures, and with low energy consumption. We hope that this enables neuromorphic systems to find applications in real-time vision processing with tight constraints on energy consumption and response times.

Appendix A

Abbreviations

Throughout the thesis, we use the following mathematical symbols:

r	spike rate of regular frequency
ν	spike rate of Poisson frequency
V	Voltage or membrane potential
p	Probability distribution, transition probability in a Markov chain
P	Probability, with the special case:
$\mathrm{P}(\mu, n)$	Poisson distribution
t,T	time or time interval
n, m, p, i	number or index of spikes
N, k	number or index of neurons
d	difference of the center of two Gauss distributions with standard deviation σ

The following abbreviations are used:

CAVIAR	*C*onvolution *A*ER (Address-Event-Representation) *V*ision *A*rchitecture for *R*eal-time, EC 5th framework project IST-2001-34124
VLSI	*V*ery *L*arge *S*cale *I*ntegration
aVLSI	analog *V*ery *L*arge *S*cale *I*ntegration
MATLAB	mathematical program of Mathworks, Inc.
D/A	digital-to-analog (converter)
RAM	random-access memory
INI	Institute of Neuroinformatics, Uni-ETH Zurich, Switzerland
IMSE	Institute of Micoelectronics, Sevilla, Spain
UIO	University of Oslo
USE	niversity of Sevilla
USB	Universal serial bus
PCI	Peripheral components interface
AER	address-event representation

Appendix B

Chip Documentation

B.1 Biases

Table B.1: Chip biases, 1st version (tsmcneuroncaviar)

Bias name	V_{def}	V_{off}	type	Comment	Pin
General biases					
Follbias	0.70V	0.00V	N	sets bias for follower in pads	9
nsrcfb0	0.84V	0.00V	N	sets voltage for source driver in scanner output	5
nsrcfb1	0.42V	0.00V	N	sets voltage for source driver in scanner output	6
pdbiasS	0.70V	0.00V	N	needed to start clocking in scanner	0
psrcfbias	2.29V	3.30V	P	should be matched with nsrcf0	8
Vcas_e	2.00V	3.30V	P	cascode	0
Vcas_i	1.00V	0.00V	N	cascode	7
Neuron Soma					
NeuronInput_3	3.30V	3.30V	P	sets DC input current to neuron	5
NeuronInputInh	3.30V	3.30V	P	sets DC input current to global inh neuron 2	2
NeuronInputInh2	3.30V	3.30V	P	sets DC input current to global inh neuron 1	1

(continued on next page)

Table B.1: Chip biases, 1st version (tsmcneuroncaviar), *continued*

Bias name	V_{def}	V_{off}	type	Comment	Pin
NeuronLeak	0.00V	0.00V	N	sets leak current of neuron	0
NeuronPul	2.40V	3.30V	P	sets rise time of discharging membrane pulse	7
NeuronPw	2.25V	3.30V	P	sets pulse width of output spike	8
NeuronRefractBias	0.63V	0.00V	N	sets refractory period	4
NeuronResetV	0.00V	0.00V	N	sets reset voltage for neuron	6
NeuronRp	0.98V	0.00V	N	sets bias current of threshold comparator	5
NeuronVcas_ngleak	0.00V	1.50V	N	sets cascode transistor of second leak transistor to membrane potential	0
NeuronVcas_pgleak	3.30V	2.50V	P	sets cascode transistor of second leak mechanism to second leak transistor	9
NeuronVgleak	0.00V	0.40V	N	sets conductance of second leak mechanism	1
NeuronVt	2.20V	0.00V	N	sets threshold of Vm before spike occurs	6
tau_gainleak	0.00V	0.40V	N	sets source node of aerisyn and second leak mechanism ratio of this and tau_leak sets final current in second leak transistor	4
tau_leak	0.40V	0.40V	N	sets source node of leak transistor	2
NeuronAdapTauSrc	0.00V	0.00V	N	sets gain in adaptation	1
NeuronAdaptBias	3.30V	3.30V	P	sets charge dumped per spike in adaptation	3
NeuronCascAdapt	0.00V	0.00V	N	sets cascode bias in adaptation	9
AER and AER Synapses					
ae_dw	0.01V	0.00V	N	set depression rate in depressing synapse	5
ae_tau	3.19V	3.30V	P	sets bias in aer exc synapse	7
ai_tau	0.15V	0.00V	N	sets gain in aer inh synapse	2

(continued on next page)

Table B.1: Chip biases, 1st version (tsmcneuroncaviar), *continued*

Bias name	V_{def}	V_{off}	type	Comment	Pin
aiw_3	2.00V	3.30V	P	sets synaptic weight for nondac inh AER synapse	1
chipreqbias	2.37V	3.30V	P	bias to set delay for req before addresses are valid	1
dacsettlingbias	0.46V	0.00V	N	bias to set delay for dac to settle before acknowledging	0
pdbiasR	0.65V	0.00V	N		2
pdbiasTX	0.85V	0.00V	N		9
pubiasTX	1.93V	3.30V	P		8
Pwbias	1.99V	3.30V	P	generates fixed pw in aer pulse	8
rpd_tau	0.00V	0.00V	N	sets synaptic weight of depressing synapse	6
SynPubias	2.00V	3.30V	P	needed to generate ack to aer circuit	1
tau_dacaew	0.19V	0.00V	N	sets Vgs across synapse weight of nFET in exc synapse	9
tau_dacaiw	2.73V	3.30V	P	sets Vgs across synapse weight of pFET in inh synapse use same value of Vdd10	0
Vcm_nfet	0.65V	0.50V	N	sets bias current for nFET source follower bias in aerisyn	7
Vcm_pfet	2.10V	3.00V	P	sets bias current for pFET source follower bias in aeresyn	8
D/A converters					
Dacpd	0.00V	0.00V	N	shared pin of masterbias tie to gnd (pulse to get it going)	5
DacVdd10	3.30V	3.30V	P	vdd10 power supply for dac output	0
DacVg	1.20V	0.00V	N	bias for amplifier of dac	8
Vdcdac1_3	0.00V	0.00V	N	sets dc current for dacaiw	3
Vdcdac2_3	0.00V	0.00V	N	sets dc current for dacaew	4

(continued on next page)

Table B.1: Chip biases, 1st version (tsmcneuroncaviar), *continued*

Bias name	V_{def}	V_{off}	type	Comment	Pin
Local Connectivity					
lself	0.00V	0.00V	N	sets synaptic weight for local exc synapse	3
tau_le	3.30V	3.30V	P	Vdd	4
tau_li	0.15V	0.00V	N	Gnd sets gain in local inh synapse	3
Winner-take-all					
l4iw	0.00V	0.00V	N	sets synaptic weight for neuron to drive inh neuron 1	6
li2iexcw	0.00V	0.00V	N	sets exc syn wt from 4 array inh neurons to global inh neuron 2	1
li2iinhw	3.30V	3.30V	P	sets inh syn wt from global neuron 1 to global inh neuron 2	2
li4w1	3.30V	3.30V	P	sets synaptic weight for inh neuron 1 to neuron	4
li4w2	3.30V	3.30V	P	sets synaptic weight for global inh neuron 2 to neuron	5
liglobal	0.00V	0.00V	N	sets inh syn wt from spglobalin to global neuron 2	3
spglobalpwbias	3.30V	3.30V	P	sets pulse width of spglobalin signal for competition across chips	2

Table B.2: Chip biases, 2nd version (tncb)

Bias name	V_{def}	V_{off}	type	Comment	Pin
General biases					
Follbias	0.70V	0.00V	N	sets bias for follower in pads	4
nsrcfb0	0.84V	0.00V	N	shared with pdbiasTX. Sets voltage for source driver in scanner output	3

(continued on next page)

Table B.2: Chip biases, 2nd version (tncb), *continued*

Bias name	V_{def}	V_{off}	type	Comment	Pin
nsrcfb1	0.42V	0.00V	N	sets voltage for source driver in scanner output	4
pdbias	0.70V	0.00V	N	needed to start clocking in scanner	1
pdbiasR	0.63V	0.00V	N		7
psrcfbias	2.29V	3.30V	P	should be matched with nsrcf0	6
pubiasR	1.90V	3.30V	P		6
pubiasTX	1.80V	3.30V	P		8
vcas_e	2.00V	3.30V	P	cascode	7
vcas_inh	1.00V	0.00V	N	cascode	6
Neuron Soma					
NeuronInput	2.70V	3.30V	P	sets DC input current to neuron	3
NeuronInput2	3.30V	3.30V	P	sets DC input current 2 to neuron	5
NeuronInputInh	3.30V	3.30V	P	sets DC input current to global inh neuron 2	3
NeuronInputInh2	3.30V	3.30V	P	sets DC input current to global inh neuron 1	2
NeuronLeak	0.00V	0.00V	N	sets leak current of neuron	6
NeuronLeak2	0.00V	0.00V	N	sets leak current 2 of neuron	4
NeuronPul	2.51V	3.30V	P	sets rise time of discharging membrane pulse	9
NeuronPw	2.01V	3.30V	P	sets pulse width of output spike	8
NeuronRefractBias	0.30V	0.00V	N	sets refractory period	2
NeuronResetV	0.43V	0.00V	N	sets reset voltage for neuron	4
NeuronRp	0.70V	0.00V	N	sets bias current of threshold comparator	1
NeuronVt	1.60V	0.00V	N	sets threshold of Vm before spike occurs	0
NeuronAdapTauSrc	0.00V	0.00V	N	sets gain in adaptation	5
NeuronAdaptBias	3.30V	3.30V	P	sets charge dumped per spike in adaptation	3
NeuronCascBias	0.00V	0.00V	N	sets cascode bias in adaptation	7

(continued on next page)

Table B.2: Chip biases, 2nd version (tncb), *continued*

Bias name	V_{def}	V_{off}	type	Comment	Pin
AER and AER synapses					
ae_dw	0.00V	0.00V	N	set depression rate in synapse	8
ae_tau	3.30V	3.30V	P	sets gain in aer exc synapse	9
aew	0.60V	0.00V	N	sets synaptic weight for exc AER synapse	0
ai_tau	0.00V	0.00V	N	sets gain in aer inh synapse	8
aiw	2.60V	3.30V	P	sets synaptic weight for inh AER synapse	7
Pwbias	2.24V	3.30V	P	generates fixed pw in aer pulse	1
rpd_tau	1.10V	3.30V	P	sets synaptic weight of depressing synapse	2
SynPubias	2.30V	3.30V	P	needed to generate ack to aer circuit	2
Local Connectivity					
lself	0.00V	0.00V	N	sets synaptic weight for local exc synapse	0
tau_le	3.30V	3.30V	P	Vdd	6
tau_li	0.00V	0.00V	N	Gnd sets gain in local inh synapse	9
tau_Vs	0.50V	3.30V	P	sets drain node of src follower transistor driven by Vbi/Vbe	0
Vbe	3.30V	3.30V	P	sets synaptic weight (value is Vbe + Vdd-Vleake) of local excitatory synapse	9
Vbi	3.30V	3.30V	P	sets synaptic weight (value is Vbi + Vdd-Vleaki) of local inhibitory synapse	1
Vleak_e	1.85V	2.80V	P	sets time constant of local excitatory synapse	8
Vleak_i	1.87V	2.80V	P	sets time constant of local inhibitory synapse	2

(continued on next page)

Table B.2: Chip biases, 2nd version (tncb), *continued*

Bias name	V_{def}	V_{off}	type	Comment	Pin
Winner-take-all					
l4iw	0.00V	0.00V	N	sets synaptic weight for neuron to drive inh neuron 1	7
li2iexcw	0.00V	0.00V	N	sets exc syn wt from 4 array inh neurons to global inh neuron 2	5
li2iinhw	3.30V	3.30V	P	sets inh syn wt from global neuron 1 to global inh neuron 2	3
li4w1	3.30V	3.30V	P	sets synaptic weight for inh neuron 1 to neuron	9
li4w2	3.30V	3.30V	P	sets synaptic weight for global inh neuron 2 to neuron	8
liglobal	0.00V	0.00V	N	sets inh syn wt from spglobalin to global neuron 2	4
pulselength	3.30V	3.30V	P	to set length of pulse for spglobal in	5

Table B.3: Chip biases, 3rd version (tnc3)

Bias name	V_{def}	V_{off}	type	Comment	Pin
General biases					
Follbias	0.70V	0.00V	N	sets bias for follower in pads	9
nsrcfb0	0.84V	0.00V	N	sets voltage for source driver in scanner output	5
nsrcfb1	0.42V	0.00V	N	sets voltage for source driver in scanner output	6
pdbiasS	0.70V	0.00V	N	needed to start clocking in scanner	0
psrcfbias	2.29V	3.30V	P	should be matched with nsrcf0	8
Vcas_e	2.00V	3.30V	P	cascode	0
Vcas_i	1.00V	0.00V	N	cascode	7

(continued on next page)

Table B.3: Chip biases, 3rd version (tnc3), *continued*

Bias name	V_{def}	V_{off}	type	Comment	Pin
Neuron Soma					
NeuronInput_3	3.30V	3.30V	P	sets DC input current to neuron	5
NeuronInputInh	3.30V	3.30V	P	sets DC input current to global inh neuron 2	2
NeuronInputInh2	3.30V	3.30V	P	sets DC input current to global inh neuron 1	1
NeuronLeak	0.00V	0.00V	N	sets leak current of neuron	0
NeuronPul	2.40V	3.30V	P	sets rise time of discharging membrane pulse	7
NeuronPw	2.25V	3.30V	P	sets pulse width of output spike	8
NeuronRefractBias	0.63V	0.00V	N	sets refractory period	4
NeuronResetV	0.00V	0.00V	N	sets reset voltage for neuron	6
NeuronRp	0.98V	0.00V	N	sets bias current of threshold comparator	5
NeuronVcas_ngleak	0.00V	1.50V	N	sets cascode transistor of second leak transistor to membrane potential	0
NeuronVcas_pgleak	3.30V	2.50V	P	sets cascode transistor of second leak mechanism to second leak transistor	9
NeuronVgleak	0.00V	0.40V	N	sets conductance of second leak mechanism	1
NeuronVt	2.20V	0.00V	N	sets threshold of Vm before spike occurs	6
tau_gainleak	0.00V	0.40V	N	sets source node of aerisyn and second leak mechanism ratio of this and tau_leak sets final current in second leak transistor	4
tau_leak	0.40V	0.40V	N	sets source node of leak transistor	2
NeuronAdapTauSrc	0.00V	0.00V	N	sets gain in adaptation	1
NeuronAdaptBias	3.30V	3.30V	P	sets charge dumped per spike in adaptation	3
NeuronCascAdapt	0.00V	0.00V	N	sets cascode bias in adaptation	9

(continued on next page)

Table B.3: Chip biases, 3rd version (tnc3), *continued*

Bias name	V_{def}	V_{off}	type	Comment	Pin
AER and AER Synapses					
ae_dw	0.01V	0.00V	N	set depression rate in depressing synapse	5
ae_tau	3.19V	3.30V	P	sets bias in aer exc synapse	7
ai_tau	0.15V	0.00V	N	sets gain in aer inh synapse	2
aiw_3	2.00V	3.30V	P	sets synaptic weight for nondac inh AER synapse	1
chipreqbias	2.37V	3.30V	P	bias to set delay for req before addresses are valid	1
dacsettlingbias	0.46V	0.00V	N	bias to set delay for dac to settle before acknowledging	0
pdbiasR	0.65V	0.00V	N		2
pdbiasTX	0.85V	0.00V	N		9
pubiasTX	1.93V	3.30V	P		8
Pwbias	1.99V	3.30V	P	generates fixed pw in aer pulse	8
rpd_tau	0.00V	0.00V	N	sets synaptic weight of depressing synapse	6
SynPubias	2.00V	3.30V	P	needed to generate ack to aer circuit	1
tau_dacaew	0.19V	0.00V	N	sets Vgs across synapse weight of nFET in exc synapse	9
tau_dacaiw	2.73V	3.30V	P	sets Vgs across synapse weight of pFET in inh synapse use same value of Vdd10	0
Vcm_nfet	0.65V	0.50V	N	sets bias current for nFET source follower bias in aerisyn	7
Vcm_pfet	2.10V	3.00V	P	sets bias current for pFET source follower bias in aeresyn	8
D/A converters					

(continued on next page)

Table B.3: Chip biases, 3rd version (tnc3), *continued*

Bias name	V_{def}	V_{off}	type	Comment	Pin
Dacpd	0.00V	0.00V	N	shared pin of masterbias tie to gnd (pulse to get it going)	5
DacVdd10	3.30V	3.30V	P	vdd10 power supply for dac output	0
DacVg	1.20V	0.00V	N	bias for amplifier of dac	8
Vdcdac1_3	0.00V	0.00V	N	sets dc current for dacaiw	3
Vdcdac2_3	0.00V	0.00V	N	sets dc current for dacaew	4
Local Connectivity					
lself	0.00V	0.00V	N	sets synaptic weight for local exc synapse	3
tau_le	3.30V	3.30V	P	Vdd	4
tau_li	0.15V	0.00V	N	Gnd sets gain in local inh synapse	3
Winner-take-all					
l4iw	0.00V	0.00V	N	sets synaptic weight for neuron to drive inh neuron 1	6
li2iexcw	0.00V	0.00V	N	sets exc syn wt from 4 array inh neurons to global inh neuron 2	1
li2iinhw	3.30V	3.30V	P	sets inh syn wt from global neuron 1 to global inh neuron 2	2
li4w1	3.30V	3.30V	P	sets synaptic weight for inh neuron 1 to neuron	4
li4w2	3.30V	3.30V	P	sets synaptic weight for global inh neuron 2 to neuron	5
liglobal	0.00V	0.00V	N	sets inh syn wt from spglobalin to global neuron 2	3
spglobalpwbias	3.30V	3.30V	P	sets pulse width of spglobalin signal for competition across chips	2

B.2 Chip Interface PCBs

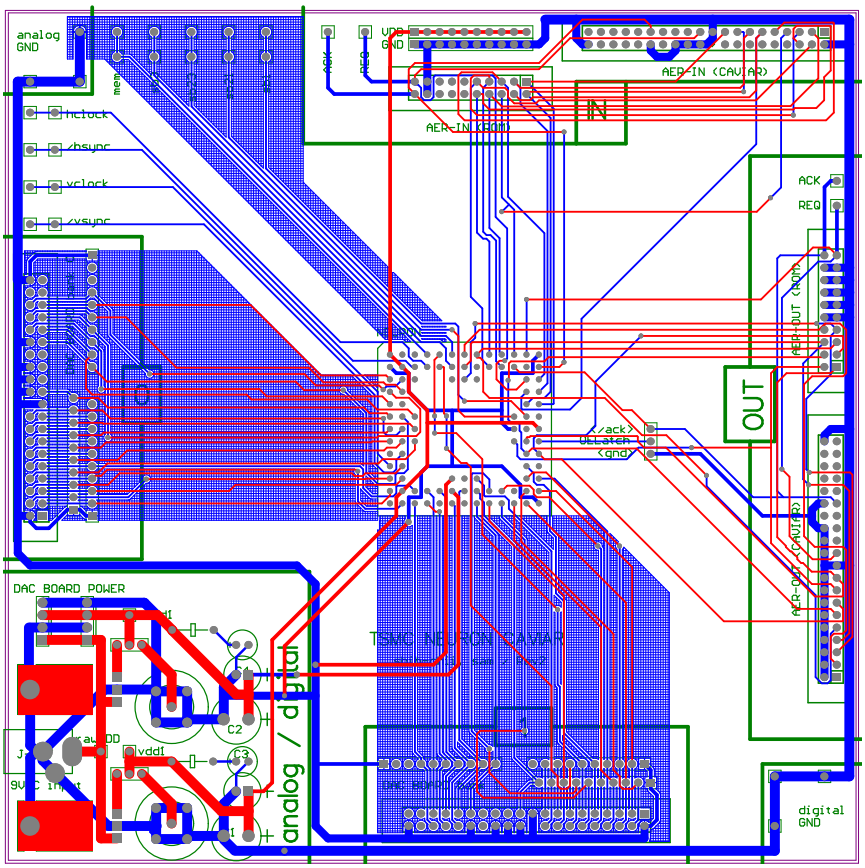

Figure B.1: Chip interface PCB, 1st version (tsmcneuroncaviar). Top view. The power supply (bottom left) generates the analog and digital supply voltages for the chip (middle). Two connectors for dacboards (bottom and left) supply the bias voltages. AER input (top) and output (right) have both CAVIAR and 20-pin standard connectors. Test connectors allow access to dedicated test pins (top left), to the biases and the /REQ and /ACK lines. The jumper at the right side selects between the point-to-point and the SCX version of the CAVIAR protocol. On this board the unused output address pins are grounded, violating the SCX standard.

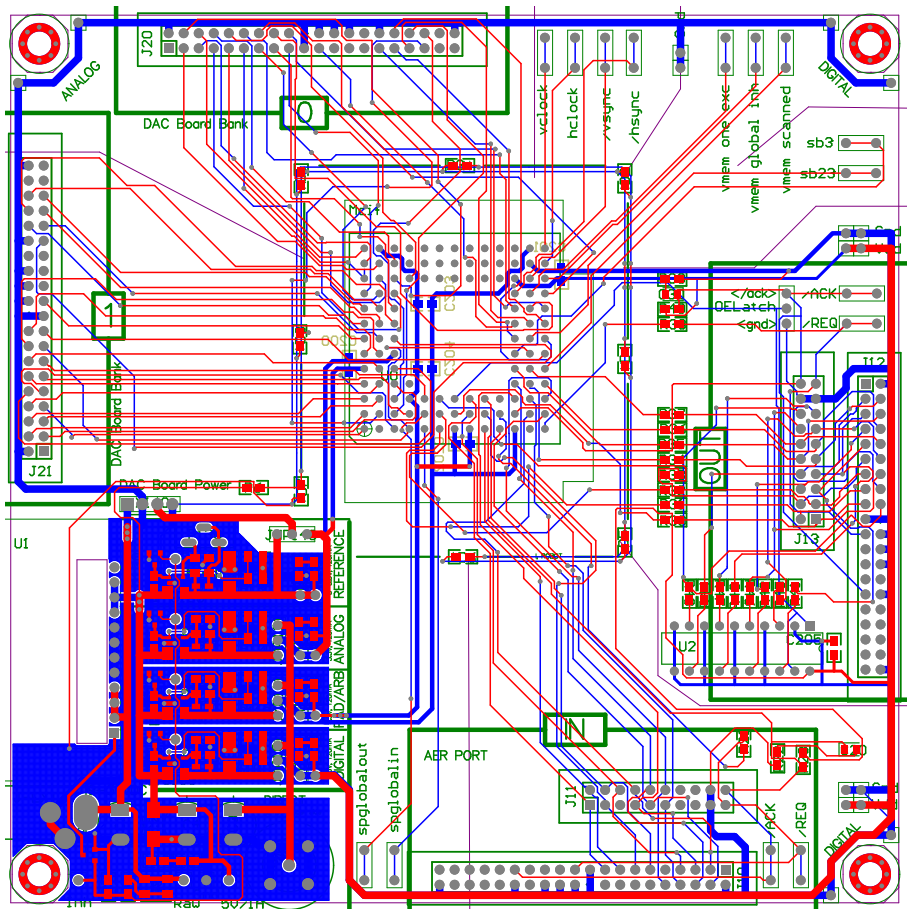

Figure B.2: Chip interface PCB, 2nd version (tsmcneuroncaviarb). Top view. In addition to analog and digital voltages, the power supply (bottom left) also generates a PAD/Arbiter supply and a reference voltage for the dacboards which allows to raise the biase voltages above 3.3V to complete shut off the P-type transistors. Digital and analog signals are seperated as much as possible (bottom right / top left part). The unused address lines are now buffered with external tri-state buffer chips. Resistor pads could be used to suppress reflections on the output lines (not used). The ring of pads around the socket is used for light-emitting diodes that provide a blue illumination of the chip socket without special functionality.

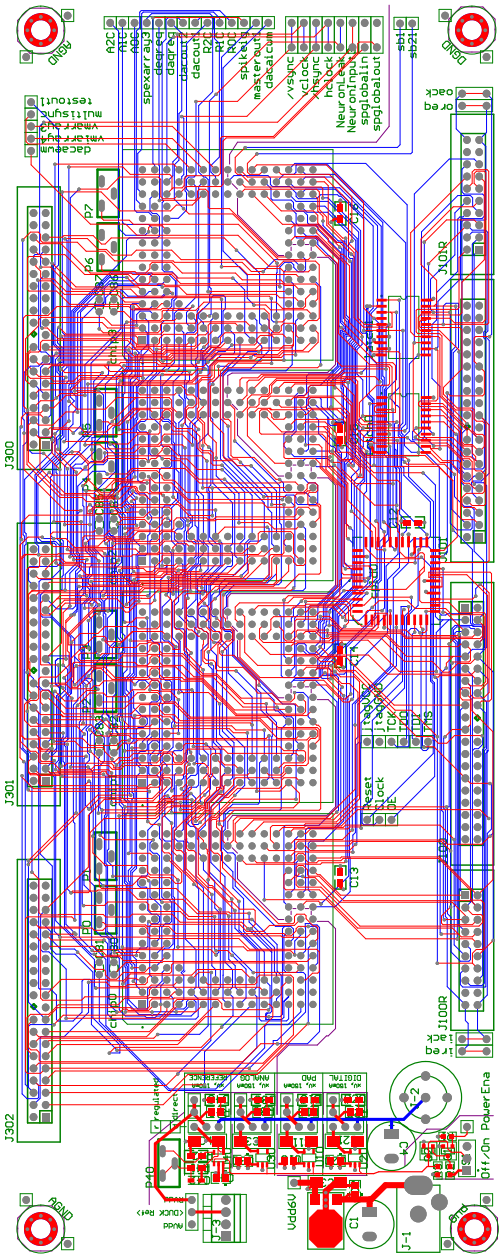

Figure B.3: Chip interface PCB, 3rd version (tnc3). Top view. *(continued on next page)*

Figure B.3 (*continued from last page*): Chip interface PCB, 3rd version (tnc3). Top view. The board can hold four chips in parallel to enable the across-chip competition. Analog (left) and digital part (right) are completely separated and feature internal split planes with one layer for ground and power. The power pins are connected through these internal power planes. The chips share the largest part of their biases, except a set of four biases that can be set individually for each chip. The biases are supplied by the 3 dacboard connectors on the left. The AER buses are merged onto a common bus (right), using a small CPLD and a set of buffers. Only the test pins of the top-most chip are instrumentized for measurement (top). We had problems with shorts on the board due to errors in the manufacturing. With more than one chip the board starts to go into power oscillations (see page 70).

Appendix C

ChipDatabase

The *ChipDatabase* is a graphical system to set biases on neuromorphic analog VLSI chips. It defines the functionality of the chip pinout and how the pins are connected to digital/analog computer interfaces. It creates a graphical user interface in MATLAB to provide an intuitive method for tuning the biases of neuromorphic chips. It facilitates the exchange of aVLSI chips by defining a common interface, allowing remote tuning and the sharing of bias settings over the web.

Neuromorphic aVLSI chips have been under investigation by a community of researchers for a long time. Although continuous development has brought single chips to a highly advanced level, the technology did not yet attract a large number of applications, and the use by other people than the original chip designers remains difficult. Setup and operation of the chips requires too much skill and experience; more that can be expected from a novice end user. To some degree the reason for this lays in the complexity of the analog design, but more importantly here, many of the chips are missing basic documentation and a convenient way of operating them. The biases have to be tuned to a narrow operating range - and these settings are easily lost if the setup changes and difficult to retrieve at a later time.

To push the development of neuromorphic systems to a scale that can deal with real-world problems, the community relies on exchange and cooperation between different labs. Much effort has been put in to facilitate the communication between the researchers themselves, for example at the annual Telluride workshop or by the Institute of Neuromorphic Engineering (INE). On a level of the spiking hardware, the European Union has started an extensive effort to develop common standards (AER) with its two CAVIAR and ALAVLSI projects in the life-like perception initiative. However, the exchange of the analog chips remains difficult due to the lack of a defined biasing interface.

This ChipDatabase project addresses these problems by

- Creating a *graphical user interface (GUI)* that allows the user to intuitively

tune an analog VLSI chip by setting biases from the standard MATLAB working environment, without knowing about the underlying hardware interfaces.

- Defining a standardized *documentation* of chips, adapter boards and setup that includes names instead of cryptic pin numbers, a description of the bias functionality, default voltages etc., all together in a flexible, easy-to-use and extendable database standard.

- The use of computerized D/A hardware, controlled by a high-level mathematical language, allows an easy and complete *characterization* of the chips. The same environment, together with data acquisition systems, can be used for *remote tuning* of chips if they are interchanged between different labs.

All these possibilities come at a price, of course. A lot of information has to be entered before the GUI can be used. However, it can also be seen as good that the designer is forced to document his chip in a common standard for the sake of easy operability. Also, the database distinguishes between the definition of the chip itself and the test setup including adapter boards etc, which are normally built by different developers.

This project developed from a quick hack of software that was done in one evening. While it gained in functionality, it also suffered from an increased complexity due to the different needs when more people started using it. Nevertheless, the development has lead to a stable current version.

The ChipDatabase is used in most of the aVLSI systems at the Institute of Neuroinformatics (INI) and in the Sensory Integration Project (INE) at universities in Tucson, AZ and Edinburgh. Usage started on the 'dacboard' developed in the CAVIAR project as the underlying D/A hardware system, and is pursuit on its successors, the 'duckboard' and the 'AMPA' board. The Sensory Integration Project used a dedicated board with additional functionality for data acquisition and processing and is weight-optimized for a flying system [Conradt, 2005]. The software package was presented in a short note with the SensoryIntegration board in the Neuromorphic Engineering newsletter [Oster, 2005].

Figure C.1 gives an overview of the software architecture. First, all specifications are entered into the database. Section C explains the format and the required data fields. The database is used by the MATLAB code to create a graphical user interface that gives the user an intuitive control over the chip biases (see section C). The database is parsed to translate function calls using chip and bias names to the information about physical device and channel that is sent to the hardware driver. In addition, bias settings, GUI layouts and other newly created information is saved to local files that can be merged into the database.

Database Format The database is based on the Extendable Markup Language (XML) [Bray et al., 2006], since it offers a versatile and hierarchical format that can be easily extended. The files can be edited with a standard text editor

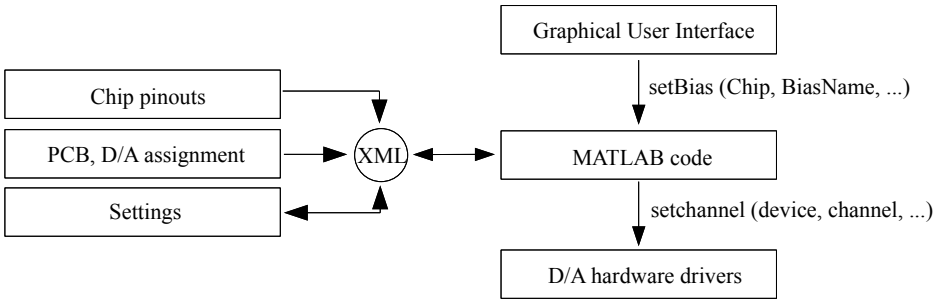

Figure C.1: Architecture Overview: All specifications (chip pinouts, PCB and D/A assignments etc) are parsed into a XML database file. The MATLAB code uses this database to translate functions with chip/bias names, issued by the user, into the physical device/channel information that is sent to the hardware driver. Additionally, bias settings are stored back to the database.

and displayed in any newer browser. The syntax is easily understandable by anyone who is familiar with the concepts of HTML. Most important, MATLAB offers an easy way to import and process XML documents in a built-in java class that complies with the Document Object Model (DOM). See [Apparao et al., 1998] for a reference. Although undocumented by MATLAB itself, all information can be inferred from the DOM standard and standard `method()` calls on the data structure.

An entry in XML format consists of *tags* and *attributes* of the form

```
<tag attribute1="value1" attribute2="value2" ...>content</tag>
```

If no content is given, it can be shortened to

```
<tag attribute1="value1" attribute2="value2" .../>
```

Hierarchical structures can be built by nesting tags as content into other tags. Files can include arbitrary tags and attributes that are ignored during further processing, so any kind of additional content can be added. The order of the entries is not important.

In the database directory, a separate file is used for each item such as a chip pinout, a PCB assignment, settings etc. The DOM requires that all data is encapsulated in a single file, so the separate XML files are parsed into one file before being loaded into MATLAB (`chipdatabase.xml`). The parsing is controlled by a `Makefile`. In addition, there is a perl script that builds an XML file from text files that are easier to create. The files are differentiated by their ending, for example `.chipclass` denotes an unparsed text file for a chipclass definition, `.chipclass.xml` the parsed XML file. The `Makefile` command `make` automatically takes care of the necessary parsing if the ending is known. All

`*.xml` are then concatenated into the `chipdatabase.xml` file. The command `make clean` removes all generated files. If there are `.xml` files from a different source, they are automatically included if they have a known ending, and they are not removed by the `make clean` command.

The following database items are currently supported by the database:

chipclass contains all information about a chip. It is a class since a setup could contain several chips of the same type. Beside the chip pinout, additional information such as default voltages, logical groups, etc. are included for each bias. The following tags with their attributes are necessary to create a fully functional graphical user interface (GUI) in MATLAB:

<`chipclass`>: defines the chipclass with the name `id`. Contains the other tags as content.

<`vdd`>: Supply voltage.

<`package`>: Package code to automatically generate a graphical pinout.

<`signal`>: Signal (pin) definition with the following attributes:

id	Name of the signal.
pin	Number of the pin on the package.
group	Logical group of the signal. Biases belonging to one group are sorted into one window by the GUI.
type	Type of the signal: bias (`bias`), analog output (`aout`), digital pin (`dout`, `din`) or power connection (`power`). The following fields apply only to the type bias:
biastype	Type of the transistor (`N` or `P`). Determines the direction of the slider in the GUI.
defaultvoltage	Default voltage.
offvoltage	Voltage if the bias is switched off. If not given, it is derived from the biastype.
comment	Description of the bias that is displayed as a tool-tip in the GUI.

board defines the Printed Circuit Board (PCB) that holds the chip and connects its pins to the channels of the D/A boards. For the database, we are only interested in this pin assignment, that means which pin of the chip is connected to which channel (and type) of the D/A card. The following tags are necessary:

<`board`>: defines the board with the name `id`. Contains the other tags as content.

<`pin`>: defines a pin assignment with the following attributes:

id	Number of the pin on the package.
bank	Bank location (if several 'dacboards' or subdevices are used).
channel	Channel number in that bank.
type	Type of the signal. To avoid confusion, the type here is referenced as seen from the D/A board, in contrast to the chipclass where it is referenced as seen from the chip. The possible types are: A/D channel (`adc`), D/A channel (`dac`) or digital channel (`dout` or `din`).

setup describes all elements that are used in a test setup: the chips that are used, with unique names, and which PCB connects them to which D/A boards. In contrast to the two definition files described before, it requires a hierarchical structure and has to be created directly in XML.

`<setup>`**:** defines the setup with the name `id`. It lists the chips used in this setup:

`<chip id="" chipclass="" board="">`**:** chip with an unique name (`id`) and the given chipclass and board definition. The chip is connected to the dacboard defined by:

`<bank id="" dacboardid=""/>`**:** The `bank` corresponds to the bank field specified in the board definition. The `dacboardid` is a unique identifier of each 'dacboard'.

This information together with the corresponding `chipclass` and `board` definitions completely specifies the test setup.

setting stores a bias vector, for example a working setup, that was saved from within MATLAB.

guilayout contains a saved layout of the graphical user interface, also created by MATLAB.

The `chipclass` and `board` definitions can be generated with a helper script from an intermediate format.

Matlab User Interface Once all necessary information is parsed into the ChipDatabase, it is loaded into MATLAB by the command

```
getSetup ('setupname')
```

which loads the database into a java class and selects a setup. If a file `chipdatabase.xml` exists in the current directory, it is loaded. Otherwise, the database is loaded from the distribution directory.

The procedure does not have a return value, but creates two global variables:

rootnode The XML document containing the root XML node as specified by the DOM. This node contains the complete database.

setupnode A pointer to the node in the XML tree that contains the setup with the specified name.

Note that these variables stay in the memory until they are explicitly cleared. If one specifies a different setting without first clearing the rootnode, the old XML database is used, but a warning will be issued. The built-in command `xmlwrite` can be used to output the XML tree for debug purposes.

The MATLAB ChipDatabase code uses the database information to translate the names for the chips and biases into hardware driver calls. If the user or the GUI issue the command

```
setBias ('ChipName','BiasName', value)
```

MATLAB parses the XML tree to get the right device and channel number. Also, the bias voltage is referred to the chip Vdd supply voltage since the D/A converters use values relative to their scale and resolution. See table C.1 for an illustration of the complete algorithm. Finally the command

```
setchannel (dacboardid, type, channel, value)
```

is called on the underlying hardware driver of the D/A board.

```
setBias (chipname, biasname, value)
```

search	`chipname`	in	`setupnode`:
get	`chipclass` name		
get	`board` name		
get	`vdd`	in	`chipclass` node
search	`biasname`	in	`chipclass` node:
get	`pin` number		
search	`pin` number	in	`board` node:
get	`bank`		
get	`type`		
get	`channel`		
search	`bank`	in	`setup` node
get	`dacboardid`		

```
setchannel (dacboardid, type, channel, value/vdd)
```

Table C.1: Algorithm to translate chip/bias names to device/channel commands.

The command

```
guiCreate
```

starts the graphical user interface (GUI). For each bias group defined in the `chipclass`, a separate window is created with the chip and group name in the title (see Figure C.2). It contains the bias names and a slider to graphically set the bias value. The N or P marker determines the direction in which the voltage increases. The text field lists the value that is currently set. If the *off button* is checked, the bias is set to the 'off' value. This is the `offvoltage` field if specified in the chipclass. If no off voltage is specified, GND is used for a N type bias, VDD for a P type. When the button is pressed a second time, the voltage that was active when the bias was switched off is restored. The last button is a push button to set the current voltage to the predefined *default voltage* from the chipclass definition.

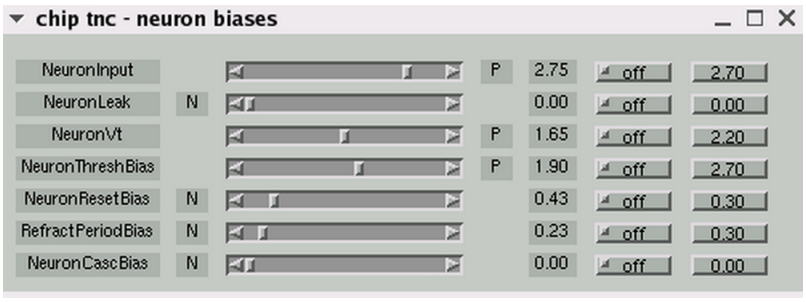

Figure C.2: Example bias group window of the Graphical User Interface

The current tuning is stored with the command

```
storeSetting ('name')
```

This creates a file called `name.setting.xml` in the current directory, which contains the biases for all chips in the current setup.

The ChipDatabase code contains several helper and library commands that can be issued by the user. The most important are:

`setBias ('ChipName', 'BiasName', value):` sets a bias value directly without using the GUI. The command is mostly used from within scripts. `ChipName` can be omitted if there is only one chip in the current setup. Currently, the GUI is not updated (see the command `updateBias` for this). The new bias value is given in Volts.

`value = getBias ('ChipName', 'BiasName'):` equivalent function to retrieve the bias value that is currently set.

`savetoeeproms:` stores the values that are currently set in the EEPROM memory of the 'dacboard', so they are automatically loaded when the 'dacboard' is powered up the next time. This prevents setting P biases to ground after a power failure.

Hardware Driver functions The CAVIAR and ALAVLSI projects use the 'dacboard' hardware while the Sensory Integration project uses a dedicated interface board. The functionality of these different boards is hidden from the database code by different hardware drivers (`DriverDACBoard` and `DriverBlimpCommands`). These drivers encapsulate any low-level communication functions that are dependent on the operating system (for example, `DriverBlimpCommands` access different `.mex` code on Windows and Linux machines). Which driver is used, is determined by the MATLAB path: The GUI calls the following commands which have to be supplied by the drivers:

```
setchannel (dacboardid, channel, value, type)

value = getchannel (dacboardid, channel, type)

setchannels (dacboardid, values, type)

values = getchannels (dacboardid, type)
```

The division into device descriptors, subdevice types and channel numbers complies with the standards set up by the `comedi` project, which supports drivers for a selection of data acquisition cards. If new hardware is to be integrated into the system, it would be desirable to include a generic interface to the standardized `comedi` functions.

Bibliography

L.F. Abbott and S.B. Nelson. Synaptic plasticity: taming the beast. *Nature Neuroscience*, 3:1178–1183, November 2000.

L.F. Abbott, J.A. Varela, K. Sen, and S.B. Nelson. Synaptic Depression and Cortical Gain Control. *Science*, 275(5297):221–224, 1997.

J.P. Abrahamsen, P. Hafliger, and T.S. Lande. A time domain winner-take-all network of integrate-and-fire neurons. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 5, 2004.

V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. Le Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, and L. Wood. W3C recommendation: Document object model level 1, October 1998.

J.V. Arthur and K. Boahen. Recurrently connected silicon neurons with active dendrites for one-shot learning. In *IEEE International Joint Conference on Neural Networks*, volume 3, 2004.

A. Bandyopadhyay, P. Hasler, and D. Anderson. A CMOS floating-gate matrix transform imager. *Sensors Journal*, 5(3):455–462, 2005.

J.A. Barnden and K. Srinivas. Temporal winner-take-all networks: a time-based mechanism for fast selection in neural networks. *IEEE transactions on neural networks*, 4(5):844–853, 1993.

C. Bartolozzi and G. Indiveri. A neuromorphic selective attention architecture with dynamic synapses and integrate-and-fire neurons. In *Proceedings of Brain Inspired Cognitive Systems*, 2004.

R. Berner. Highspeed USB2.0 AER interfaces. Master's thesis, Institute of Neuroinformatics, UNI-ETH Zurich and Architectura y Technologia de Computadores, Universidad de Sevilla, 2006.

T. Binzegger, R. Douglas, and K. Martin. A quantitative map of the circuit of cat primary visual cortex. *Journal of Neuroscience*, 24(39):8441–53, 2004.

K.A. Boahen. Point-to-point connectivity between neuromorphic chips using address-events. *IEEE Transactions on Circuits & Systems II*, 47(5):416–434, 2000.

K.A. Boahen. A burst-mode word-serial address-event link-I: transmitter design. *IEEE Transactions Circuits & Systems I*, 51(7):1269–1280, 2004a.

K.A. Boahen. A burst-mode word-serial address-event link-II: receiver design. *IEEE Transactions Circuits & Systems I*, 51(7):1281–1291, 2004b.

K.A. Boahen. A burst-mode word-serial address-event link-III: analysis and test results. *IEEE Transactions Circuits & Systems I*, 51(7):1292–1300, 2004c.

M. Boegerhausen, P. Suter, and S.-C. Liu. Modeling short-term synaptic depression in silicon. *Neural Computation*, 15(2):331–348, Feb 2003.

A. Bofill-i-Petit and A.F. Murray. Synchrony detection by analogue VLSI neurons with bimodal STDP synapses. In *Advances in Neural Information Processing Systems (NIPS)*, volume 16, 2003.

T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau. W3C recommendation: Extensible markup language (XML) 1.0, August 2006. Fourth Edition.

E.N. Brown, R. Barbieri, V. Ventura, R.E. Kass, and L.M. Frank. The time-rescaling theorem and its application to neural spike train data analysis. *Neural Computation*, 14(2):325–346, 2002.

L. Carota, G. Indiveri, and V. Dante. A software–hardware selective attention system. *Neurocomputing*, 58(60):647–653, 2004.

E. Chicca, G. Indiveri, and R.J. Douglas. An event-based VLSI network of integrate-and-fire neurons. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 5, 2004.

E. Chicca, P. Lichtsteiner, T. Delbruck, G. Indiveri, and R.J. Douglas. Modeling orientation selectivity using a neuromorphic multi-chip system. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, 2006.

T.Y.W. Choi, B.E. Shi, and K. Boahen. An ON–OFF Orientation Selective Address Event Representation Image Transceiver Chip. *IEEE Transactions on Circuits and Systems I*, 51(2):342–353, 2004.

T.Y.W. Choi, P.A. Merolla, J.V. Arthur, K.A. Boahen, and B.E. Shi. Neuromorphic implementation of orientation hypercolumns. *IEEE Transactions on Circuits and Systems I*, 52(6):1049–1060, 2005.

J. Conradt. Helping neuromorphic sensors leave the designer's desk. In *The Neuromorphic Engineer Newsletter*, volume 2, pages 8–9. Institute of Neuromorphic Engineering (INE), Mar 2005.

USB-If Consortium. Universal serial bus revision 2.0 specification, April 2000.

R. Coultrip, R. Granger, and G. Lynch. A cortical model of winner-take-all competition via lateral inhibition. *Neural Networks*, 5(1):47–54, 1992.

E. Culurciello and A.G. Andreou. A comparative study of access topologies for chip-level address-event communication channels. *IEEE Transactions on Neural Networks*, 14(5):1266–1277, 2003.

V. Dante and P. Del Giudice. The PCI-AER interface board. In A. Cohen, R. Douglas, T. Horiuchi, G. Indiveri, C. Koch, T. Sejnowski, and S. Shamma, editors, *2001 Telluride Workshop on Neuromorphic Engineering Report*, pages 99–103, 2001.

P. Dayan and L.F. Abbott. *Theoretical neuroscience: computational and mathematical modeling of neural systems*. MIT Press, 2001.

S.R. Deiss, R.J. Douglas, and A.M. Whatley. *A pulse-coded communications infrastructure for neuromorphic systems*, pages 157–178. MIT Press Cambridge, MA, USA, 1999.

T. Delbruck and P. Lichtsteiner. Fully programmable bias current generator with 24 bit resolution per bias. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2006.

A. Delorme. Early Cortical Orientation Selectivity: How Fast Inhibition Decodes the Order of Spike Latencies. *Journal of Computational Neuroscience*, 15(3):357–365, 2003.

A. Delorme, J. Gautrais, R. Rullen, and S.J. Thorpe. SpikeNET: A simulator for modeling large networks of integrate and fire neurons. *Neurocomputing*, 26(27):989–96, 1999.

A. Delorme, L. Perrinet, and S.J. Thorpe. Network of integrate-and-fire neurons using rank order coding b: spike timing dependant plasticity and emergence of orientation selectivity. *Neurocomputing*, 38:539–545, 2001.

C. Diorio, P. Hasler, A. Minch, and C.A. Mead. A single-transistor silicon synapse. *IEEE Transactions on Electron Devices*, 43(11):1972–1980, 1996.

R.J. Douglas and K.A.C. Martin. Neuronal Circuits of the Neocortex. *Annual Review of Neuroscience*, 27(1):419–451, 2004.

R.J. Douglas, K.A.C. Martin, and D. Whitteridge. A canonical microcircuit for neocortex. *Neural Computation*, 1:480–488, 1989.

R.J. Douglas, C. Koch, M. Mahowald, K.A. Martin, and H.H. Suarez. Recurrent excitation in neocortical circuits. *Science*, 269(5226):981, 1995.

P.G. Drennan and C.C. McAndrew. Understanding MOSFET mismatch for analog design. *IEEE Journal of Solid-State Circuits*, 38(3):450–456, 2003.

M. Eisele. *Einfluss von Parameterschwankungen auf die Ausbeute digitaler Niedervoltschaltungen*. PhD thesis, Institute for Technical Electronics, Technical University Munich, 1998.

B. Ermentrout. Complex dynamics in winner-take-all neural nets with slow inhibition. *Neural Networks*, 5(3):415–431, 1992.

G. Fuhrmann, I. Segev, H. Markram, and M. Tsodyks. Coding of Temporal Information by Activity-Dependent Synapses. *Journal of Neurophysiology*, 87(1):140–148, 2002.

K. Fukushima. Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

J. Gautrais and S.J. Thorpe. Rate coding versus temporal order coding: a theoretical approach. *Biosystems*, 48(1):57–65, 1998.

W. Gerstner and W.M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

R. Hahnloser, R. Sarpeshkar, M. Mahowald, R.J. Douglas, and S. Seung. Digital selection and analog amplification coexist in an electronic circuit inspired by neocortex. *Nature*, 405:947–951, 2000.

R. Hahnloser, H.S. Seung, and J.J. Slotine. Permitted and forbidden sets in symmetric threshold-linear networks. *Neural computation*, 15(3):621–638, 2003.

P. Hasler, P. Smith, C. Duffy, C. Gordon, J. Dugger, and D. V. Anderson. A floating-gate vector-quantizer. In *IEEE Midwest Circuits and Systems*, Tulsa, OK, Aug 2002.

Y. He and E. Sanchez-Sinencio. Min-net winner-take-all CMOS implementation. *Electronics Letters*, 29(14):1237–1239, 1993.

C.M. Higgins, V. Pant, and R. Deutschmann. Analog VLSI implementation of spatio-temporal frequency tuned visual motion algorithms. *IEEE Transactions on Circuits and Systems*, 52(3):489–502, 2005.

D.H. Hubel and T.N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of Physiology*, 160(1): 106–154, 1962.

G. Indiveri. Winner-Take-All Networks with Lateral Excitation. *Analog Integrated Circuits and Signal Processing*, 13(1):185–193, 1997.

G. Indiveri. A Current-Mode Hysteretic Winner-take-all Network, with Excitatory and Inhibitory Coupling. *Analog Integrated Circuits and Signal Processing*, 28(3):279–291, 2001.

G. Indiveri, A.M. Whatley, and J. Kramer. A reconfigurable neuromorphic VLSI multi-chip system applied to visual motion computation. In *Proceedings of the Seventh International Conference on Microelectronics for Neural, Fuzzy and Bio-inspired Systems*, pages 37–44, 1999.

G. Indiveri, T. Horiuchi, E. Niebur, and R. Douglas. A competitive network of spiking VLSI neurons. In *World Congress on Neuroinformatics*, pages 443–455, 2001.

G. Indiveri, P. Oswald, and J. Kramer. An adaptive visual tracking sensor with a hysteretic winner-take-all network. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 2, 2002.

L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, 1998.

B.J. Jain and F. Wysotzki. Discrimination networks for maximum selection. *Neural Networks*, 17(1):143–54, 2004.

D.Z. Jin and H.S. Seung. Fast computation with spikes in a recurrent neural network. *Physical Review E*, 65:051922, May 2002.

S. Kaski and T. Kohonen. Winner-take-all networks for physiological models of competitive learning. *Neural Networks*, 7(6-7):973–984, 1994.

R.E. Kass, V. Ventura, and E.N. Brown. Statistical issues in the analysis of neuronal data. *Journal of Neurophysiology*, 94(1):8–25, July 2005.

T. Kincaid, M.A. Cohen, and Y. Fang. Dynamics of a Winner-Take-All Neural Network. *Neural Networks*, 9(7):1141–1154, 1996.

C. Koch and I. Segev. The role of single neurons in information processing. *Nature Neuroscience*, 3:1171–1177, 2000.

E. Korner, M. Gewaltig, U. Korner, A. Richter, and T. Rodemann. A model of computation in neocortical architecture. *Neural Networks*, 12:989–1005, 1999.

J. Kramer. An integrated optical transient sensor. *IEEE Transactions on Circuits and Systems II, Analog and Digital Signal Processing*, 49(9):612–628, Sep 2002.

J. Lazzaro, S. Ryckebusch, M.A. Mahowald, and C.A. Mead. Winner-Take-All circuits of O(n) complexity. In *Advances in Neural Information Processing Systems (NIPS)*, volume 1, pages 703–711, 1989.

D.D. Lee and H.S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

P. Lichtsteiner, C. Posch, and T. Delbruck. A 128x128 120dB 30mW asynchronous vision sensor that responds to relative intensity change. In *ISSCC Digest of Technical Papers*, pages 508–509, 2006.

A. Linares-Barranco, M. Oster, D. Cascado, G. Jiménez, A. Civit, and B. Linares-Barranco. Inter-spike-intervals Analysis of Poisson Like Hardware Synthetic AER Generation. *Lecture notes in computer science*, pages 479–485, 2006.

B. Linares-Barranco, T. Serrano-Gotarredona, and R. Serrano-Gotarredona. Compact Low-Power Calibration Mini-DACs for Neural Arrays With Programmable Weights. *IEEE Transactions on Neural Networks*, 14(5):1207, 2003.

B. Linares-Barranco, T. Serrano-Gotarredona, R. Serrano-Gotarredona, and J. Costas-Santos. A new charge-packet driven mismatch-calibrated integrate-and-fire neuron for processing positive and negative signals in AER based systems. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 5, pages 744–747, 2004.

S.-C. Liu. A neuromorphic aVLSI model of global motion processing in the fly. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 47(12):1458–1467, 2000a.

S.-C. Liu. A normalizing aVLSI network with controllable winner-take-all properties. *Analog Integrated Circuits and Signal Processing*, 31(1):47–53, 2002.

S.-C. Liu. A Winner-Take-All circuit with controlable Soft Max property. In *Advances in Neuron Information Processing (NIPS)*, pages 717–723, 2000b.

S.-C. Liu and M. Oster. Feature competition in a spike-based winner-take-all VLSI network. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 3634–3637, May 2006.

S.C. Liu, J. Kramer, G. Indiveri, T. Delbruck, T. Burg, and R. Douglas. Orientation-selective aVLSI spiking neurons. *Neural Networks*, 14(6):629–643, 2001.

J. Louie. A biological model of object recognition with feature learning. Master's thesis, Artificial Intelligence Lab MIT, 6 2003.

E.D. Lumer. Effects of Spike Timing on Winner-Take-All Competition in Model Cortical Circuits. *Neural Computation*, 12:181–194, 2000.

W. Maass. Neural computation with winner-take-all as the only nonlinear operation. In *Advances in Neural Information Processing Systems (NIPS)*, volume 12, 1999.

W. Maass. On the Computational Power of Winner-Take-All. *Neural Computation*, 12:2519–2535, 2000.

W. Maass, T. Natschläger, and H. Markram. Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations. *Neural Computation*, 14:2531–2560, 2002.

W. Maass, T. Natschlager, and H. Markram. *Computational models for generic cortical microcircuits*, chapter 18, pages 575–605. Chapman & Hall/CRC, Boca Raton, 2004.

K. Madani, G. de Trémiolles, and P. Tannhof. Image Processing Using RBF like Neural Networks: A ZISC-036 Based Fully Parallel Implementation Solving Real World and Real Complexity Industrial Problems. *Applied Intelligence*, 18(2):195–213, 2003.

U. Mallik, R.J. Vogelstein, E. Culurciello, R. Etienne-Cummings, and G. Cauwenberghs. A Real-time Spike-Domain Sensory Processing System. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 3, page 1919. IEEE; 1999, 2005.

H. Markram, Y. Wang, and M. Tsodyks. Differential signaling via the same axon of neocortical pyramidal neurons. *PNAS*, 95(9):5323–5328, 1998.

C. Mead. *Analog VLSI and neural systems*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.

S. Mitra, S. Fusi, and G. Indiveri. A VLSI spike-driven dynamic synapse which learns. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2777–2780. IEEE, May 2006.

R. Moller. A strong winner-take-all neural network in analogue hardware. *Neuromorphic Systems: Engineering Silicon from Neurobiology. World Scietic*, 1998.

A. Mortara and E.A. Vittoz. A communication architecture tailored for analog VLSI artificial neural networks: intrinsic performance and limitations. *IEEE Transactions on Neural Networks*, 5(3):459–466, 1994.

A. Mortara, E.A. Vittoz, and P. Venier. Communication scheme for analog VLSI perceptive systems. *IEEE Journal of Solid-State Circuits*, 30(6):660–669, 1995.

M. Ogawa, K. Ito, and T. Shibata. A general-purpose vector-quantization processor employing two-dimensional bit-propagating winner-take-all. In *Symposium on VLSI Circuits*, pages 244–247, 2002.

G. Ortelli. Two analog vlsi motion sensors for flight stabilization. Semester project, Institute of Neuroinformatics, Zurich, 2004.

M. Oster. Tuning aVLSI chips with a mouse click. In *The Neuromorphic Engineer Newsletter*, volume 2, page 9. Institute of Neuromorphic Engineering (INE), 2005.

M. Oster and S.-C. Liu. Interface 'object' chip with computer. CAVIAR Deliverable Workpackage 3.6, 2004. IST-2001-34124.

M. Oster, A.M. Whatley, S.-C. Liu, and R.J. Douglas. A Hardware/Software Framework for Real-Time Spiking Systems. In *Lecture Notes in Computer Science*, volume 3696, page 161. Springer, 2005.

E. Ozalevli and C.M. Higgins. Reconfigurable biologically inspired visual motion systems using modular neuromorphic VLSI chips. *IEEE Transactions on Circuits and Systems*, 52(1):79–92, 2005.

L. Perrinet, A. Delorme, M. Samuelides, and S.J. Thorpe. Networks of integrate-and-fire neuron using rank order coding A: How to implement spike time dependent Hebbian plasticity. *Neurocomputing*, 38:817–822, 2001.

J. Postel. Transmission control protocol, rfc 761. USC/Information Sciences Institute, January 1980a.

J. Postel. User datagram protocol, rfc 768. USC/Information Sciences Institute, August 1980b.

D.S. Reich, F. Mechler, and J.D. Victor. Formal and Attribute-Specific Information in Primary Visual Cortex. *Journal of Neurophysiology*, 85(1):305–318, 2001.

F. Rieke, D. Warland, R. Steveninck, and W. Bialek. *Exploring the Neural Code*. MIT press, Cambridge, 1996.

M. Riesenhuber and T. Poggio. Models of object recognition. *Nature Neuroscience*, pages 1199–204, 2000.

M. Riesenhuber and T. Poggio. Neural mechanisms of object recognition. *Current Opinion in Neurobiology*, 12(2):162–8, 2002.

M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–25, 1999.

H.K. Riis and P. Hafliger. Spike based learning with weak multi-level static memory. In *International Symposium on Circuits and Systems (ISCAS)*, volume 5, 2004.

M. Rivas, F. Gomez-Rodriguez, R. Paz, A. Linares-Barranco, S. Vicente, and D. Cascado. Tools for Address-Event-Representation Communication Systems and Debugging. In *Lecture Notes In Computer Science*, volume 3696, page 289. Springer-Verlag, 2005.

A. Rodríguez-Vázquez, G. Liñán, S. Espejo, and R. Domínguez-Castro. Mismatch-Induced Trade-Offs and Scalability of Analog Preprocessing Visual Microprocessor Chips. *Analog Integrated Circuits and Signal Processing*, 37(2):73–83, 2003.

E. Salinas and L.F. Abbott. A model of multiplicative neural responses in parietal cortex. *PNAS*, 93(21):11956–11961, 1996.

T. Serrano, B. Linares-Barranco, and N.M. Center. A modular current-mode high-precision winner-take-all circuit. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 5, 1994.

R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vincent, F. Gòmez-Rodríguez, H. Kolle Riis, T. Delbrück, S.-C. Liu, S. Zahnd, A.M. Whatley, R.J. Douglas, P. Häfliger, G. Jimenez-Moreno, A. Civit, T. Serrano-Gotarredona, A. Acosta-Jiménez, and B. Linares-Barranco. AER building blocks for multi-layer multi-chip neuromorphic vision systems. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 18, pages 1217–1224, Cambridge, MA, Dec 2005. Neural Information Processing Systems Foundation, MIT Press.

R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jimnez, and B. Linares-Barranco. A neuromorphic cortical layer microchip for spike based event processing vision systems. *IEEE Transactions on Circuits and Systems I*, 2006. To be published.

T. Serrano-Gotarredona and B. Linares-Barranco. A modified ART1 algorithm more suitable for VLSI implementations. *Neural Networks*, 9(6):1025–1043, 1996.

T. Serrano-Gotarredona and B. Linares-Barranco. An ART1 microchip and its use in multi-ART1 systems. In *Proceedings of 1997 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 1, pages 673–676, 1997.

T. Serrano-Gotarredona and B. Linares-Barranco. Systematic Width-and-Length Dependent CMOS Transistor Mismatch Characterization and Simulation. *Analog Integrated Circuits and Signal Processing*, 21(3):271–296, 1999.

T. Serrano-Gotarredona and B. Linares-Barranco. A new five-parameter MOS transistor mismatch model. *Electron Device Letters, IEEE*, 21(1):37–39, 2000.

T. Serre, M. Riesenhuber, J. Louie, and T. Poggio. On the role of object-specific features for real world object recognition in biological vision. In *Workshop on Biologically Motivated Computer Vision (BMCV)*, Nov 2002.

H.S. Shapiro and R.A. Silverman. Alias-Free Sampling of Random Noise. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):225–248, 1960.

J.A. Starzyk and X. Fang. CMOS current mode winner-take-all circuit with both excitatory and inhibitory feedback. *Electronics Letters*, 29(10):908–910, 1993.

C.F. Stevens. Quantal release of neurotransmitter and long-term potentiation. *Cell*, 72:55–63, 1993.

A.A. Stocker. Analog Integrated 2-D Optical Flow Sensor. *Analog Integrated Circuits and Signal Processing*, 46(2):121–138, 2006.

S.J. Thorpe and J. Gautrais. Rank order coding. *Computational Neuroscience: Trends in Research*, pages 113–8, 1998.

S.J. Thorpe, R. Guyonneau, N. Guilbaud, J.M. Allegraud, and R. VanRullen. SpikeNet: real-time visual processing with one spike per neuron. *Neurocomputing*, 58(60):857–864, 2004.

S. Ullman, M. Vidal-Naquet, and E. Sali. Visual features of intermediate complexity and their use in classification. *Nat Neurosci*, 5(7):682–7, 2002.

L.G. Ungerleider and J.V. Haxby. "what" and "where" in the human brain. *Current Opinion in Neurobiology*, 4(2):157–65, 1994.

R. Van Rullen and S.J. Thorpe. Rate Coding Versus Temporal Order Coding: What the Retinal Ganglion Cells Tell the Visual Cortex. *Neural Computation*, 13(6):1255–1283, 2001.

R. VanRullen and S.J. Thorpe. Surfing a spike wave down the ventral stream. *Vision research*, 42(23):2593–2615, 2002.

R.J. Vogelstein, U. Mallik, and G. Cauwenberghs. Beyond address-event communication: dynamically-reconfigurable spiking neural systems. In *The Neuromorphic Engineer*, volume 1. Institute of Neuromorphic Engineering (INE), 2004.

D.L. Wang. Object selection based on oscillatory correlation. *Neural Networks*, 12(4):579–592, 1999.

Y.-X. Wang and S.-C. Liu. Programmable synaptic weights for an aVLSI network of spiking neurons. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2006.

F. Woergoetter, A. Cozzi, and V. Gerdes. A Parallel Noise-Robust Algorithm to Recover Depth Information From Radial Flow Fields. *Neural Computation*, 11(2):381–416, 1999.

X. Xie, R. Hahnloser, and H.S. Seung. Learning winner-take-all competition between groups of neurons in lateral inhibitory networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 13, pages 350–356, 2001.

X. Xie, R. Hahnloser, and H.S. Seung. Selectively grouping neurons in recurrent networks of lateral inhibition. *Neural Computation*, 14(11):2627–2646, 2002.

M. Yagi and T. Shibata. An associative-processor-based mixed signal system for robust grayscale image recognition. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, volume 5, 2002.

M. Yagi, T. Shibata, and K. Takada. Optimizing feature-vector extraction algorithm from grayscale images for robust medical radiograph analysis. In *Proceedings of the 5th Biannual World Automation Congress*, volume 13, pages 251–257, 2002.

Z. Yang, A. Murray, F. Worgotter, K. Cameron, and V. Boonsobhak. A neuromorphic depth-from-motion vision model with STDP adaptation. *Neural Networks*, 17(2):482–495, 2006.

A.J. Yu, M.A. Giese, and T.A. Poggio. Biophysiologically Plausible Implementations of the Maximum Operation. *Neural Computation*, 14:2857–2881, 2002.

A.L. Yuille and D. Geiger. Winner-take-all mechanisms. *The handbook of brain theory and neural networks*, pages 1056–1060, 1998.

A.L. Yuille and N.M. Grzywacz. A winner-take-all mechanism based on presynaptic inhibition feedback. *Neural Computation*, 1:334–347, 1989.

Acknowledgements

This work was funded by the EC 5th framework project CAVIAR (IST-2001-34124), the INE project 'SensoryIntegration' and the Institute of Neuroinformatics (INI), Uni-ETH Zurich.

We would like to acknowledge Vittorio Dante and Paolo Del Giudice (Istituto Superiore di Sanità, Rome, Italy) for the original design of the PCI-AER board, and Adrian Whatley, Gerd Dietrich and all other members of the Institute of Neuroinformatics UNI-ETH Zurich involved in the development of the PCI-AER board, of its drivers, and software library components.

I like to express my thanks to the following people: my supervisor Shih-Chii Liu for letting me develop my own ideas and still guiding me with her way of doing research; Rodney Douglas for his encouraging and clear opinions; together with Kevan Martin for running the INI as the lively place for science and building personality that it is; Bernabe Linares-Barranco for managing CAVIAR and being my coreferent; Anton Civit for his cordial invitation to spend two weeks on the project in Sevilla; Chuck Higgins for his hospitality in Arizona; Patrick Lichtsteiner for our fun in traveling to conferences around the world; all my colleagues, mentioning especially Adrian Whatley, Anita Schmidt, Chiara Bartolozzi, Connie Hofstötter, Daniel D. Rubin, Henning Proske, Jakob Heinzle, Jason Rolfe, Jörg Conradt, Jörg Hipp, Milanka Stankovic, Nuno da Costa, Pratap Kumar, Samuel Zahnd and Tobi Delbruck for their advice, help and many things more, and Wolfgang Einhäuser for bringing me to the field of Neuroinformatics.

The CAVIAR crew in picture:

Curriculum Vitae

Personal

Address:
Institute of Neuroinformatics
Winterthurerstr. 190
CH-8057 Zurich, Switzerland
Phone: +41 44 635 3044
Fax: +41 44 635 3053
Email: Matthias.Oster@ini.phys.ethz.ch

Date of Birth: 13.06.1977
Citizenship: German
Languages: German native, English fluent, Italian and French basic

Education

Doctoral Studies (by Nov. 9th, 2006)

2002 - 2006
Institute of Neuroinformatics, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland
PhD Thesis: "Spike-based Winner-Take-All Computation in a Multi-Chip Vision System"

Diploma in Electrical Engineering and Information Technology

1997–2002, grade 1.4 (rank 7 of 99)
Munich University of Technology (TUM), Germany
Specialization in Information and Communication Technologies
Master Thesis at the Institute for Electronic Design Automation: "Estimating the performance of analog circuit components using Fourier-Motzkin-elimination"

Theresiengymnasium München, Germany

1987 - 1996, Abitur grade 1.1 (best of class)
Specialization in Mathematics and Greek

Publications

Peer-reviewed Conference Papers

Liu, S.-C. and Oster, M.: Feature competition in a spike-based winner-take-all VLSI network, In *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2006

Oster, M and Liu, S.-C.: Spiking Inputs to a Winner-take-all Network, In *Advances in Neural Information Processing Systems (NIPS)*, 2005

Serrano-Gotarredona, R. and Oster, M. et al.: AER Building Blocks for Multi-Layer Multi-Chip Neuromorphic Vision Systems, In *Advances in Neural Information Processing Systems (NIPS)*, 2005

Oster, M., Whatley, A. M., Liu, S.-C. and Douglas, R. J.: A Hardware/Software Framework for Real-time Spiking Systems, In *15th International Conference on Artificial Neural Networks (ICANN)*, 2005

Oster, M. and Liu, S.-C.: A Winner-take-all Spiking Network with Spiking Inputs, In *11th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2004

Others

Oster, M. et al: CAVIAR - Building Blocks for a Large-Scale Spiking Vision System, *50th Anniversary Summit of Artificial Intelligence*, 2006

Oster, M.: Tuning aVLSI chips with a mouse click, *The Neuromorphic Engineer Newsletter, 2005*

Ad Hoc Networking - Technology and Trends, *CDTM Trend Report 2000/2001*, BoD Norderstedt, 2001

Experience

Teaching

Workshop "The brain from the toolbox", CdE Academy, Bad Hersfeld, 2006
Co-Supervision of semester project of Gregorio Ortelli, INI 2004
Teaching Assistant for the Institute for Real Time Computer Systems (2001), Institute for Measurement Systems and Sensor Technology (1999/2000) and the Institute for Flight Mechanics (1999), TUM

Management

Venture Challenge for young entrepreneurs, VentureLab Zurich, 2006
System administration and network management at the Telluride Neuromorphic Workshop, 2005
Certificate of the Center for Digital Technology and Management (CDTM), 2001

Industry and Internships

Infineon Technologies AG / Project Racing Group, U.S.

Aug. - Oct. 2000 /April 2001
Data analysis and running the data acquisition systems in the CART racing car series, at the race tracks in the U.S, Canada and Australia

Knorr Bremse Munich, Germany

June - July 1997
Metal processing, included in the apprenticeship training

Collaborations and Projects

CAVIAR - Convolution Address-Event-Representation Vision Architecture for Real-Time

2002 - 2006
EU project IST-2001-34124
Bernabe Linares-Barranco, Instituto de Microelectrónica de Sevilla (IMSE), Consejo Superior de Investigaciones Científicas (CSIC), Spain
Philipp Häfliger, Institutt for Informatikk, Universitetet i Oslo, Norway
Anton Civit, Arquitectura y Tecnología de Computadores, Universidad de Sevilla, Spain

Sensory integration on a flying robot

2004
INE collaboration (Institute of Neuromorphic Engineering, Maryland, U.S.)
Charles Higgins, Neuromorphic Vision System Lab, Univ. of Arizona, U.S.,
Barbara Webb (Univ. of Edinburgh, Scotland)

Scholarships and Awards

International Conference on Artificial Neural Networks (ICANN), student travel grant, 2005

Studienstiftung des deutschen Volkes (German National Academic Foundation), 2001-2002
e-fellows.net, 2001-2005
Bundeswettbewerb Mathematik, prices in 1991-1996
Deutsche SchülerAkademie, 1994

Paper Referee

IEEE Transactions on Neural Networks (TNN), IEEE Press
IEEE International Symposium on Circuits and Systems (ISCAS)