1-1-2007

An adaptive, self-organizing, neural wireless sensor network.

Matthew Ball
University of Windsor

AN ADAPTIVE, SELF-ORGANIZING, NEURAL WIRELESS SENSOR NETWORK

by

Matthew Ball

A Thesis
Submitted to the Faculty of Graduate Studies
through Electrical Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada

2007

Your file Votre référence
ISBN: 978-0-494-35180-2
Our file Notre référence
ISBN: 978-0-494-35180-2

Abstract

Networking and processing software intended for wireless sensor networks must achieve energy-efficient, fault-tolerant processing while simultaneously tolerating node mobility, enduring frequent node failures, and providing sufficient flexibility to allow for deployment in a wide variety of applications.

A new architecture is proposed that simultaneously achieves network self-organization and neural processing through a unified approach. The recurrent self-organizing map forms the basis of the architecture's neural behavior, providing the necessary temporal sensitivity that allows the network to analyze a wide range of real-world, time-varying signals. The proposed architecture is designed with consideration for common phenomena related to wireless sensor networks, such as random deployment, node mobility, and resource scarcity.

Simulations demonstrate the architecture's tolerance of node mobility, resilience against noise, and self-organizing behavior.

Dedication

To my parents, for their limitless support and encouragement.

Acknowledgements

I would like to express my most sincere gratitude to my supervisor, Dr. Jonathan Wu. My academic achievements throughout graduate school are owed to his continual encouragement and guidance. I must also thank Dr. Abdul-Fattah Asfour, for encouraging me to pursue graduate education and offering years of valuable advice and support. I also wish to express my thanks to Ms. Andria Turner, Mr. Frank Cicchello, and all the departmental staff for their hard work, dedication, and eagerness to lend a helping hand to a student such as myself.

Contents

List of Figures

List of Abbreviations

ACE:	Algorithm for Cluster Establishment
ANN:	Artificial Neural Network
CCR:	Corner-Cube Retroreflector
DARPA:	Defense Advanced Research Projects Agency
DSN:	Distributed Sensor Networks
DSR:	Dynamic Source Routing
ESF:	European Science Foundation
GAF:	Geographically Adaptive Fidelity
HCP:	Hexagonal Close-Packing
HEED:	Hybrid Energy-Efficiency Distributed Clustering
IPTO:	Information Processing Techniques Office, DARPA
LEACH:	Low Energy Adaptive Clustering Hierarchy
MEMS:	Micro Electromechanical Systems
QoS:	Quality of Service
RSOM:	Recurrent Self-Organizing Map
SINA:	Sensor Information Networking Architecture
SNR:	Signal-to-Noise Ratio
SOM:	Self-Organizing Map
SOSUS:	Sound Surveillance System
WSN:	Wireless Sensor Network

Chapter 1

Introduction

1.1 Wireless Sensor Networks

A wireless sensor network (WSN) is a wireless network consisting of low cost, low power, multifunctional devices known as sensor nodes. Nodes are spatially distributed within a region of interest, though their precise locations need not be engineered or predetermined [1]. Node hardware typically includes a sensor, simple processing elements, and a wireless transceiver that facilitates communication within a limited radius. While the processing capability of each node is limited, the network benefits from the high degree of aggregate parallel processing resulting from the collaborative effort of many sensor nodes. WSNs have been identified as one of the most important upcoming technologies that will change the world [3]. Potential and current applications include inventory tracking [4], the monitoring of disaster areas [4, 5, 6], traffic control [2], medical monitoring [7], space and planetary exploration [7] and military surveillance [2, 4, 6].

Research leading to advances in wireless sensor network technology represents an amalgam of research efforts in the fields of sensors, communications, and computing [2]. While isolated advancements in any of these fields subsequently leads to impetus in the area of WSNs, as early as the late 1970s, research dedicated to the specific advancement of wireless sensor networks became recognized as an autonomous discipline and was

supported through funding. A principal benefactor of WSN research at this time was the Defense Advanced Research Projects Agency (DARPA) through its Distributed Sensor Networks (DSN) program [2].

1.1.1 Overview

Military applications have served as motivation for the research of many technologies, including wireless sensor networks. WSN projects conducted mostly in the United Sates, and specifically those by DARPA, shaped the early development of WSN research, essentially establishing a *de facto* definition of a wireless sensor network as, *"... a large-scale ad hoc, multi-hop, unpartitioned network of largely homogeneous, tiny, resource-constrained, mostly immobile sensor nodes that would be randomly deployed in the area of interest."* [8: pg. 1]. This characterization was based on the common goals of those projects as well as the limitations of the technology of the period. However, as research progressed and WSN technology evolved, this description became increasingly inadequate [8]. In 2004, the European Science Foundation (ESF) funded a workshop in which experts gathered to discuss important WSN topics and to coordinate related research activities in Europe. This workshop produced a more sophisticated scheme of characterizing different types of wireless sensor networks based on several criteria; deployment, mobility, form factor, heterogeneity, communication modality, network topology, coverage, connectivity, network size, and other quality of service (QoS) requirements [8].

1.1.2 Deployment

The way in which sensor nodes are deployed will vary with application. The placement of nodes may be random, or they may be deliberately placed in particular locations. Nodes may also be imbedded within the target environment – for example, being mixed into concrete or dropped into deep crevices, in which case they will subsequently be inaccessible for maintenance or recovery (possibly bringing about environmental concerns). Deployment may take the form of a one-time activity or may be an ongoing process in which additional nodes are distributed to increase sensor density in interesting regions or to take the place of old nodes that have become damaged, destroyed, or otherwise ineffective [8]. Methods of deployment can be classified as: *random vs. manual; imbedded vs. accessible; one-time vs. iterative*, and so on [see: 8].

1.1.3 Mobility

Following initial deployment, nodes may not remain in their original positions indefinitely. Some network applications rely on the movement of nodes as an essential phase of the sensing process. Sensing objectives that require the collection of data over a very large region, such as an entire planetary surface, cannot be realistically achieved with stationary nodes. A more practical solution is for each node to be mobile, so that they can move to the next area of interest after the examination of their previous locations have been completed. In this case, the mobility of nodes is a desirable feature of the network. For other applications, the movement itself may be the matter under investigation, such as when tracking objects in which nodes have been embedded.

Mobility can be achieved passively (the node is propelled by an autonomous entity), or actively (under its own power and control).

1.1.4 Form factor

The physical configuration of a sensor node can take several forms, influenced by cost, resource and size constraints. During the mid-1980s, the mobile nodes used in DARPA's DSN test bed had to be mounted to trucks due to their massive size and weight [2]. Since that time, technological advancements have allowed the gradual miniaturization of sensor nodes. Modern day research projects illustrate the reduction in scale that has become achievable through Micro Electromechanical Systems (MEMS) fabrication techniques. Researchers at University of California, Berkeley pursued a *Smart Dust* [9] project, whose sensor nodes are no larger than a few cubic millimeters.

The untethered nature of sensor nodes precludes any connection to established power grids. Nodes must therefore be entirely self-reliant on meeting their power demands, either by scavenging energy from their environment (i.e. solar cells) or by depleting an energy storage device such as a battery or fuel cell. Nodes that lack a renewable energy supply have a maximum operational lifetime that scales with the quantity of fuel at its disposal. In the case of battery-powered sensor nodes, lifetime is therefore limited by the size of the battery, which may represent nearly the entire volume of the node.

1.1.5 Heterogeneity

There are two design paradigms regarding the level of conformity between sensor nodes within a network. Homogeneous networks consist of nodes that are indistinguishable from one another; both the hardware and software of these nodes are identical. Homogeneous networks tend to be less expensive to manufacture due to the greater degree of mass production of its nodes, and also more manageable to deploy since no special attention is given to differing node species during placement. Conversely, heterogeneous networks include more than one class of node, and attempt to optimize available resources through the targeted deployment of specialized nodes to where they are most useful. For example, a heterogeneous network may incorporate nodes containing temperature and pressure sensors deployed to where both measurements are necessary, and nodes with only pressure sensors deployed to regions where temperature is unimportant. In contrast, a homogeneous network would be putting valuable temperature sensing hardware to waste in the latter area. Heterogeneous networks may require more sophisticated software than their homogeneous counterparts in order to properly utilize the unique talents of each node.

1.1.6 Communication modality

Sensor nodes can exploit several modes of wireless communication, some being more suited to particular environments than others. Ambient interference, line-of-sight requirement, power usage, as well as the cost and complexity of the corresponding transceiver hardware are the dominant considerations regarding the selection of a communication method. For example, the *Smart Dust* project described in [9] exploits

the small size and low power of laser beam communication technology, despite its reliance on there being line-of-sight between nodes. Possible methods of communication include radio, light, sound, etc.

1.1.7 Network topology

The network topology of a WSN is a mapping of the wireless connections between nodes. The simplest form of network topology (referred to as *single-hop*) arises when every node is capable of direct communication with all others without the participation of intermediaries. As circumstances that limit a node's ability to establish wireless connections arise, more complex topological patterns emerge. Topology is heavily influenced by the mode of communication employed (which defines the maximum range of connections as well as line-of-sight requirements), and also by infrastructure (which may necessitate the channeling of messages through base stations). As the combinational complexity of the mapping increases, the topology is typically characterized as a combination of more basic topologies, each of which is determined by the configuration of communicable nodes.

1.1.8 Coverage

The degree of coverage provided by a WSN is equivalent to how extensively an area is monitored by sensors [8, 10]. Since each sensor has a limited range, a relationship exists between the abundance of nodes employed and the level of coverage achieved. Coverage can vary from *sparse*, where some coordinates lack monitoring, to *dense*, where every coordinate is monitored by one sensor, to *redundant*, where several sensors monitor the

same coordinate. In some cases, the level of coverage provided is neither spatially nor temporally constant. A higher density of nodes may be deployed to the more interesting regions of the monitored environment, and the movement of mobile nodes can alter the degree of coverage over a period of time.

1.1.9 Connectivity

Connectivity characterizes a network's resilience against fragmentation into two or more isolated segments. The topology of a *connected* network is such that a path exists between any arbitrary pair of nodes, even if intermediate nodes are necessary for the path to be established.

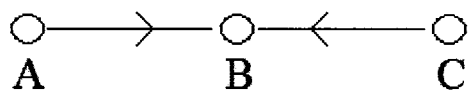

Figure 1.1: An example topology of a connected network

In Figure 1.1, three nodes form a connected network; starting at any node, a path can be found that links it to the other two. Note that the directionality of the connections is irrelevant when determining if the network is connected.

A network's *connectivity* is a quantitative attribute, equal to the minimum number of nodes that must be removed in order to create a partition in the network, thereby causing it to be disconnected. When the removal of just one node causes a partition, the network is referred to as being *1-connected.* Networks can be described as *1-connected, 2-connected, n-connected,* and so on. The topology shown in Figure 1.1 illustrates a *1-*

connected network, since the removal of a single node (node *B*) creates a partition between nodes *A* and *C*, which lack an alternate communicable path.

Connectivity reflects the robustness of communication within the network. A high level of connectivity may also bolster communication throughput by eliminating bottlenecks [12]. Connectivity, like coverage, may need to be excessive at the time of deployment, so that as nodes gradually die, connectivity will remain at a *sufficient* level.

The mobility of nodes in combination with a dynamic environment may cause temporary disconnects. When these disconnects are occasional, the connectivity is said to be *intermittent* [8]. However, as breaches in the network become more severe, and partitions result in the isolation of nodes for most of the time, connectivity is referred to as *sporadic*.

1.1.10 Energy management / lifetime

The maximization of sensor node lifetime goes hand-in-hand with the development of energy management techniques. Recall that nodes tend to be energy-constrained devices, therefore the lifetime of nodes cannot exceed that duration which the energy storage mechanism can supply power. Whereas applications may require nodes to survive for many years, special attention must be given to the issue of a node's energy efficiency, even when state-of-the-art batteries are being used.

1.1.10.1 Sleep scheduling

A common method to reduce the energy consumed by a network is to employ synchronized hibernation of sensor nodes, according to a set of predetermined scheduling

rules. During the *hibernating*, *sleep* or *idle* periods, the node's power usage is significantly reduced, while simultaneously providing little or no services to the network. To avoid interruption in the network's operation, any load born by a sleeping node must be displaced to other active nodes. In general, nodes are expected to sleep when there is little for them to do (such as when waiting for an incoming message), and the number of nodes sleeping at any time reflects the quiescence of the environment. In sleep mode, a node may turn off its radio [13], reduce power supply voltages [14], and so on.

1.1.11 Collaborative/distributed processing

Collaborative processing is an excellent feature for many distributed sensing assignments. In many cases, the data collected by the sensing network is used as a basis to form some control decision (i.e. the timing of traffic signals, the release of drugs into a patient's body, or alerting a human operator of an imminent tsunami). Though each node has limited processing hardware, the dense placement of many nodes can produce enough aggregate computational power to accomplish either the processing, or pre-processing of their sensed data locally, before sending the results to a primary control center or command station. The processing tasks can include pattern recognition, data compression, and so on. Distributed processing sensor networks offer several advantages over their sensing-only counterparts. A primary advantage is that local processing of data has an effect of reducing the number of bytes that need to be transmitted to a control station, thereby reducing communication power requirements. These energy savings are particularly significant when the transmission distance is large. Also, the network's

processing power will scale with network size; therefore the number of nodes can be adjusted to achieve an appropriate computational capability.

1.1.12 Other QoS requirements

The very wide range of applications seen by wireless sensor networks results in many unique QoS design requirements. Military applications may require nodes to be camouflaged, or entirely invisible (too small to be seen by the naked eye). Tamper-resistance and eavesdropping-resistance, as well as physical robustness are other considerations when designing individual nodes, as well as networking protocols.

1.2 Artificial Neural Networks

An artificial neural network (ANN) is a network of interconnected *artificial neurons*, modeled after biological neural networks. Neurons perform simple processes on their inputs, and forward their output along directional connections to other neurons. Since many simple interactions between neurons leads to complex global behavior, most ANNs systems give rise to emergent processes. Most ANNs are adaptive systems whose structure changes during the learning phase, in order to achieve a processing objective.

1.2.1 Supervised learning neural networks

The structural parameters of a supervised-learning neural network are updated during the learning phase by being exposed to several input/output training pairs. The learning algorithm uses the input/output pairs to infer an accurate mapping function between

them, within some small margin of error. To produce a training data set, *a priori* information about the inputs must generally be known by an external supervisor.

1.2.2 Unsupervised learning neural networks

Unlike their supervised-learning counterparts, unsupervised-learning neural networks have no *a priori* output, and no training data sets are used. Therefore, the only information available to the network during learning is the inputs themselves. As a result, unsupervised-learning neural networks *"evolve to extract features or regularities in presented patterns, without being told what outputs or classes associated with the input patterns are desired."* [33: pg. 301]. Examples of unsupervised-learning neural networks include the Self-Organizing Map (SOM), Adaptive Resonance Theory (ART) Network, and Hopfield Net.

1.2.2.1 The self-organizing map (SOM)

The self-organizing map (SOM) is a popular unsupervised-learning neural network first developed by Teuvo Kohonen in 1984, and is sometimes referred to as Kohonen's self-organizing map. An SOM is a dual-layer feed-forward neural network consisting of one input layer and one output layer. Every neuron in the output layer is linked to the multi-dimensional input by a *weight* vector of equal dimensionality (see Figure 1.2).

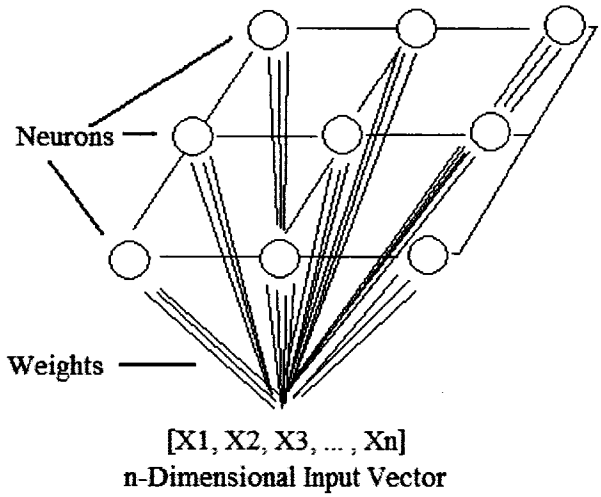

Figure 1.2: Self-organizing map structure

As shown in Figure 1.2, an *n*-dimensional input vector is connected to the output layer of neurons. Each neuron is connected to the input layer by an *n*-dimensional weight vector. Initially, each weight vector is set to small, random values. As inputs are presented to the network, the weight vectors are modified to minimize a particular error function. This has the effect of *learning* commonly encountered inputs. The SOM weight-update formula is spatially dependent, so that similar inputs will be associated with neurons in a small neighborhood, whereas very dissimilar inputs will be associated with distant neurons in the map. An SOM is therefore also referred to as a *topology-preserving feature map.*

1.3 Objectives

The objective of the research described in this thesis is to develop a simple, adaptive architecture for wireless sensor networks that takes full advantage of the hardware's massive parallelism and redundancy. Simplicity is important to facilitate implementation on limited sensor node hardware, and adaptation eliminates the lengthy and costly

process of tailoring a network to a particular task. A related objective is the seamless integration of a neural processing foundation with a wireless sensor network in order to achieve the desired parallelism in the signal processing of sensed data.

Chapter 2

Review of the State-of-the-Art

2.1 Themes in State-of-the-Art WSN Design

Issues related to the supply and consumption of power tend to dominate state-of-the-art WSN research, due to the inaccessible and energy-constrained nature of sensor nodes. While the development of superior batteries will certainly offer some relief to sensor node designers, increasing the energy efficiency of nodes will be a continuing endeavor. Wireless transceivers are the most exhausting components of sensor nodes, and most of the energy wasted by the node is associated with wireless communication. Some examples of energy waste in wireless communication include: re-transmission of lost or corrupted data packets; transmitting data that is of no interest to the receiver; generating communication signals that are stronger than is necessary to deliver data to its destination; generating communication signals that flood an area much larger than what is necessary to be received at the destination. The development of robust, multi-hop, self-organizing routing protocols is the objective of most WSN researchers as a mechanism to reduce energy wasted in these ways.

2.1.1 Multi-hop communication

Multi-hop topologies rely on the forwarding of data from its source to its destination by intermediate nodes. Multi-hop communication strategies are used to decrease the energy

overhead associated with generating wireless signals. The energy invested in the generation of an omni-directional signal scales with the square of the distance that the signal must travel. Multi-hop communication is implemented by having several intermediate nodes forward data, where each forwarding signal covers a fraction of the total difference. Figure 2.1 and Figure 2.2 illustrate single-hop and multi-hop communication strategies, respectively.

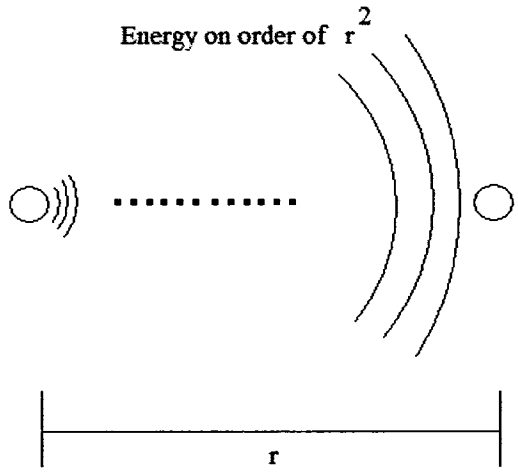

Figure 2.1: Single-hop communication

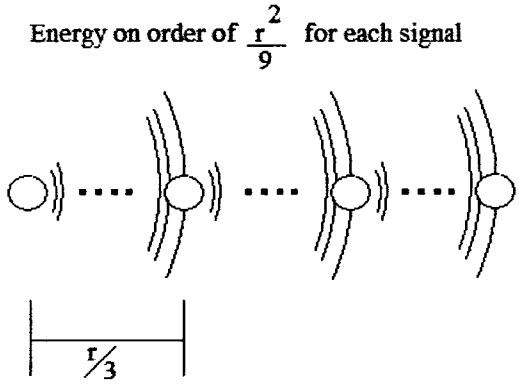

Figure 2.2: Multi-hop communication

As shown in Figure 2.2, three signals are generated, each of which only needs to travel one-third of the distance compared to the single-hop strategy shown in Figure 2.1.

Due to the exponential nature of omni-directional signal propagation, a collection of several short-range signals represent a much smaller investment of energy than one long-range signal. In the example shown in Figure 2.1 and Figure 2.2, the multi-hop strategy requires only one-third the energy of the single-hop strategy to send a signal a total distance of *r*.

2.1.2 Clustering

In many sensor network applications, sensor nodes are located close to each other, but far from base stations. In terms of energy, local communication between nodes is almost always less expensive than the long-distance communication between the nodes and the base station. A very effective and commonly implemented method to achieve significant energy savings is called *clustering*. Clustering is a category of network organization and routing that reduces the amount of data transmitted long-range by aggregating, filtering, and pre-processing data locally. Nodes must collaborate to decide which data warrants the costly transmission to the base station; nodes must also collaboratively decide which of them will generate the resultant transmission. Typically, the node aggregating the data also sends the long-distance transmission and is responsible for coordinating cluster activities. This node is referred to as the *cluster-head*, and the remaining cluster nodes are referred to as *member nodes*. Typically, all member nodes are within a single-hop of their cluster head, and are prohibited by the routing scheme from communicating with member nodes of other clusters. In addition to communicating with base stations, it is typical for cluster-heads to have the ability to communicate with each other. *Cluster overlap* occurs when a member node is within single-hop range of more than one cluster-

head. A network that suffers from a large amount of cluster overlap tends to have a higher number of total clusters, each of which has a low population. Many, low population clusters tend to be less energy efficient when compared to having fewer large population clusters.

2.2 Literature Review

University of California, Berkeley professors Kris Pister and Joe Kahn are the leaders of the DARPA-supported *Smart Dust* project detailed in [9]. Smart Dust nodes (also referred to as *motes*) were designed to have a volume no greater than one cubic millimeter ("dust"-sized), while being equipped with hardware allowing for sensing, bi-directional communication, processing, and energy generation (see Figure 2.3). The equipped solar panel was sufficient to allow for continuous power usage in the microwatt range, so the remaining hardware was designed to meet this power requirement.

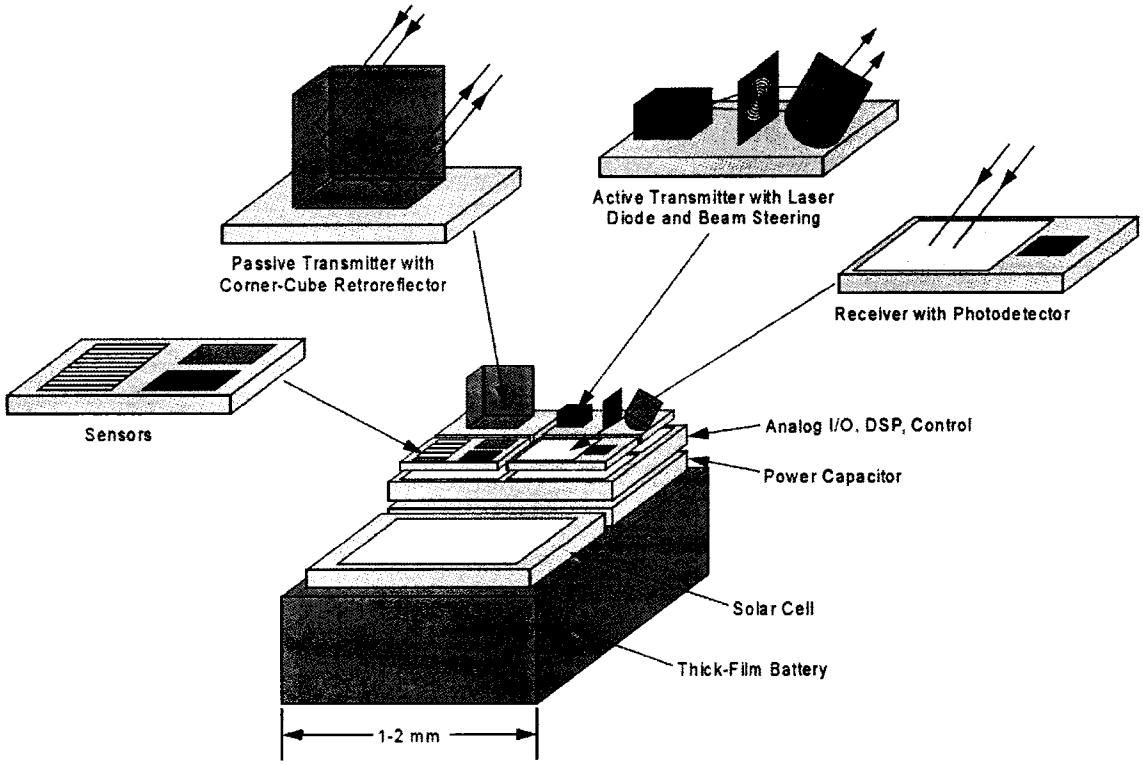

Figure 2.3: A *Smart Dust* mote, with equipped hardware (taken from [9])

Pister and his team decided that their power targets could be realized if nodes were spared the burden of generating their own communication signals. An innovative infrastructure-based communications strategy was adopted, whereby a base station would query nodes, and simultaneously supply the optical power needed by the nodes to respond. A Corner Cube Retro-reflector (CCR) was incorporated into every node in order to modulate and reflect a querying optical beam back to its originating base station while requiring almost no investment of energy by the node. Unfortunately, this requires a line-of-sight path between nodes and base stations, which need significantly more available energy than nodes. Nodes are queried not by reference to a particular node ID#, but rather spatially, since a given optical beam trajectory from the base station will only intersect a particular node.

Shen et al. developed the Sensor Information Networking Architecture (SINA) [17], which provides adaptive organization of sensor information, and facilitates query, event monitoring and tasking capability. SINA utilizes attribute-based addressing, whereby nodes are queried by the current states of several of their attributes, such as general location and the value of their sensed data. In other words, instead of the following address-based query, *"Node #127, report your data"*, the attribute-based query may take the form, *"To all nodes in the northwest region: reply if you are sensing a temperature greater than 100 degrees Celsius"*. SINA also provides clustering components to facilitate organized scalability for very large networks. A cluster is formed by collecting neighboring nodes, one of which (the *cluster head*) will take on added responsibilities of coordinating the activities of the other nodes in the collective. When appropriate, clusters themselves may be aggregated to form a cluster hierarchy.

In [4], Estrin et al. discuss the use of *directed diffusion* – a set of localized algorithms, to accomplish networking goals, rather than using a centralized approach. The authors describe localized algorithms as, *"a distributed computation in which sensor nodes only communicate with sensors within some neighborhood, yet the overall computation achieves a desired global objective"* [4: pg. 3]. It is proposed that compared to a localized approach, a centralized one is a bad choice due to inadequate scalability, energy inefficiency, and greater fragility. Their directed diffusion model is not based on a sequence of queries and replies as in other strategies, but rather each node classifies its particular attributes, and other nodes express some level of interest in those attributes. A *gradient* of interest is formed in a neighborhood of nodes, which directs the diffusion of

data from node to node. A node's attributes may include its location, the type of data being sensed (i.e. temperature, pressure, etc.), remaining energy supply, etc.

Shah and Rabaey also endorsed the idea that nodes' power supplies should be exhausted in a more uniform manner, and proposed an *Energy Aware* routing scheme in [18]. The authors criticize energy-optimizing protocols that continually rely on an *optimal* multi-hop path, since the available energy of the nodes in that path will be rapidly depleted, *"leaving the network with a wide disparity in the energy levels of the nodes, and eventually disconnected subnets"* [18: pg. 1]. Instead, the energy-aware protocols proposed by the authors are designed to maximize *network survivability*, as they define it, by maintaining networking connectivity for as long as possible. In accordance with their stated objective, Shah and Rabaey's protocol alternates between a set of *good* paths, rather than the continual reliance on an optimal path. It was shown by the authors that their energy-aware protocols could extend network lifetime by up to 40% over the directed diffusion scheme.

Low-Energy Adaptive Clustering Hierarchy (LEACH), described by Heinzelman et al. in [19], is another localized, automated clustering architecture. The authors recognized that the cluster-heads exhaust energy much more quickly than other nodes. To more evenly distribute this energy burden, the cluster-head duties are often reassigned to other nodes. The result is a more even depletion of energy supplies of all the nodes across the network, leading to a more sudden, system-wide failure, rather than a gradual crippling of the network due to an accumulation of individual node deaths over the network's entire lifetime. The former scenario is more desirable than the latter, since the network retains a high level of sensing quality until it goes offline. Also, as cluster-heads

are reassigned in LEACH, the other nodes may also be reassigned to other clusters, since the head of a different cluster may now be closer than the newly assigned cluster-head of the old cluster. The authors used simulations to compare LEACH with static clustering algorithms, and found that it took 8 times longer for the first node to die in LEACH as it does with static clustering protocols, and 3 times longer for the last LEACH node to die than the last node with static clustering protocols.

The Hybrid Energy-Efficiency Distributed Clustering (HEED) protocol described by Younis & Fahmy in [20] is a clustering protocol designed to maximize the interval between the network going online and the first node failure. The HEED protocol selects cluster-heads randomly, where nodes with more residual energy are more likely to become cluster-heads; cluster-head duties are re-assigned to other nodes at regular intervals. When two potential cluster heads have equal residual energy, the one that is associated with a lesser inter-cluster communication cost is selected. The authors reported simulation results showing minor improvements in network lifetime over the LEACH protocol.

Xu, Heidemann and Estrin described a Geographical Adaptive Fidelity (GAF) algorithm in [21]. The term, "geographically adaptive" refers to GAF's plotting of multi-hop paths between communicating nodes based on precise knowledge of their relative positions, obtained through GPS or some equivalent system. Precise geographical information can be used to ensure that the employed multi-hop path results in the least amount of wasted energy compared to alternative paths. Nodes that are not participating in the multi-hop path are powered down according to a sleep-scheduling algorithm. A load balancing strategy is also included in the GAF protocol so that nodes located in high

traffic areas do not become depleted as rapidly. The authors report that their GAF algorithm consumes 40% to 60% less energy compared to DSR (Dynamic Source Routing) [23, 22] and AODV (Ad Hoc On-Demand Distance Vector) [24, 22]. DSR and AODV are similar protocols that were developed in the 1990s for ad hoc networking of computers that are not subject to significant energy constraints (such as laptop computers).

GS^3 is a hexagonal close-packing (HCP) clustering algorithm presented by Zhang and Arora in [25]. GS^3 forms hexagonally shaped clusters starting a "big node" and spreading outwardly until the entire network is covered. Big nodes are responsible for initiating the clustering process and are the only nodes capable of communicating with external systems including base stations or other computer networks such as the Internet. In order to efficiently form hexagonal clusters with minimal overlap (overlap occurs when the boundaries of adjacent clusters intersect), GS^3 requires precise geographical information about each node, and every node in the network must lie on the same two-dimensional plane. GS^3 produces excellent cluster uniformity and less cluster overlap than competing clustering algorithms.

ACE (Algorithm for Cluster Establishment) is a clustering protocol reported by Chan and Perrig in [26] that is designed to minimize cluster overlap (overlap occurs when cluster boundaries intersect) while providing full cluster coverage. ACE is an emergent algorithm that requires no centralized authority; execution of the ACE protocol only requires nodes to communicate with neighbors within a single-hop range. ACE achieves excellent population uniformity in clusters with very little overlap. Unlike GS^3, the ACE protocol does not require geographical information about nodes. The simulations

reported in the literature show that in certain circumstances, the amount of overlap generated by ACE could approach what is achievable with Hexagonal Close-Packing algorithms such as GS^3. Recall that GS^3 is a honeycomb-packing algorithm that requires precise geographic information of each node.

Chou et al. proposed a sophisticated distributed data compression strategy to reduce communication energy in [27]. The authors suggest that there exists an inherent correlation between the sensor data acquired by densely deployed sensor nodes. The underlying correlation is the basis of consequent redundancy in the network's collective sensor data. In other words, the underlying premise of the authors' work is that *if variable Y is known, and a correlation between X and Y is known, then X must only be partially known in order to extract the full value of X as a result of X's known dependency on Y.* In essence, the strategy requests that nodes transmit data that is incomplete to varying degrees. A special "data-gathering" node can fill in the missing pieces of data by analyzing all of the incomplete fragments that it has received, assuming it knows how all of the fragments are correlated. Initially, no correlation is known, and the data-gathering node requires sensor nodes to transmit their whole readings. Correlation tracking algorithms are used by the data-gatherer to construct a correlation table between nodes. As the correlation table becomes more complete, the data-gathering node will instruct sensing nodes to transmit a smaller fraction of their data, and the correlations will be used to extract the missing pieces. This heterogeneous scheme allocates all of the intensive correlation-based computations to the data-gathering nodes. The compression algorithm used by each sensor node is rudimentary, and consists of only a single multiplication and modulo operation. The simulation results in [27] show that

the proposed strategy is robust against errors, and tolerant of noise, while achieving significant energy savings. The authors indicate that their proposed strategy can reduce the communication energy expended by each sensor node by up to 65%.

Oldewurtle and Mähönen proposed the use of a two-level, hierarchical neural network algorithm to achieve fault-tolerance, data compression and parallel processing in [28]. In [28], each sensor node would run a software-based Hopfield Net as a way to pre-process their locally collected sensor readings (the authors assumed two or more sensors would be present in each node). It must be stressed that each sensor node *does not* represent a neuron in a large Hopfield Net, but rather a virtual Hopfield Net consisting of multiple neurons is fully implemented through software at each sensor node. For example, if a sensor network consists of one hundred sensor nodes, each which has three different sensors, then there would be one hundred Hopfield Nets; each Hopfield Net is associated with a particular sensor node, and is used strictly to process the three-dimensional sensor data of the sensor node on which it is being simulated. These local Hopfield Nets form the first level of the neural network hierarchy in the literature. The second level is implemented by sensor node cluster-heads. Each sensor node transmits the output of its Hopfield Net to the cluster-head, which aggregates the data using a self-organizing map (SOM). The cluster-head uses a virtual SOM with 5x4 neurons to perform an *X-to-2* dimensionality reduction on the combined output of a cluster containing *X* sensor nodes.

The work of Kulakov et al. in [29] is similar to Oldewurtle and Mähönen [28]. A two-level hierarchy of neural networks is implemented on a WSN. Both levels of the hierarchy consist of an Adaptive Resonance Theory (ART) – style ANN. At the lower

level, each sensor node simulates a FuzzyART neural network to classify only its own sensor readings. The authors used real Smart-It sensor nodes to collect data. The Smart-It node includes six different sensors, and the virtual FuzzyART running in the node's software classifies patterns that occur in the simultaneous readings of those six sensors. The output of the FuzzyART at each node is transmitted and collected by a cluster-head. The cluster-head runs the second-level of the hierarchy, which is a binary ART network whose inputs are the FuzzyART outputs from each other node in the cluster. The architecture self-organized well and showed robustness to sensor errors.

Caterall et al. investigated the use of sensor nodes to represent individual neurons in a self-organizing map (SOM) in [30]. The physical hardware used in the author's experiment consisted of five Smart-It nodes, which each contain six different sensors. In an effort to reduce communication overhead, the authors made a slight change to the original SOM model. Recall that in a classic SOM (see Figure 1.2) a single input vector is presented to each neuron. Consequently, the input vector seen by each neuron is identical. If Caterall et al. adhered to the original SOM model, the input to each neuron would be a thirty-dimensional vector (five nodes × six sensors each). However, each node has only local access to its own six sensor readings, therefore the remaining twenty-four inputs would need to be transmitted wirelessly. In other words, whenever a node took sensor readings, it would need to wirelessly transmit that data to every other node. The authors opted to modify the original SOM and arrived at the model illustrated in Figure 2.4.

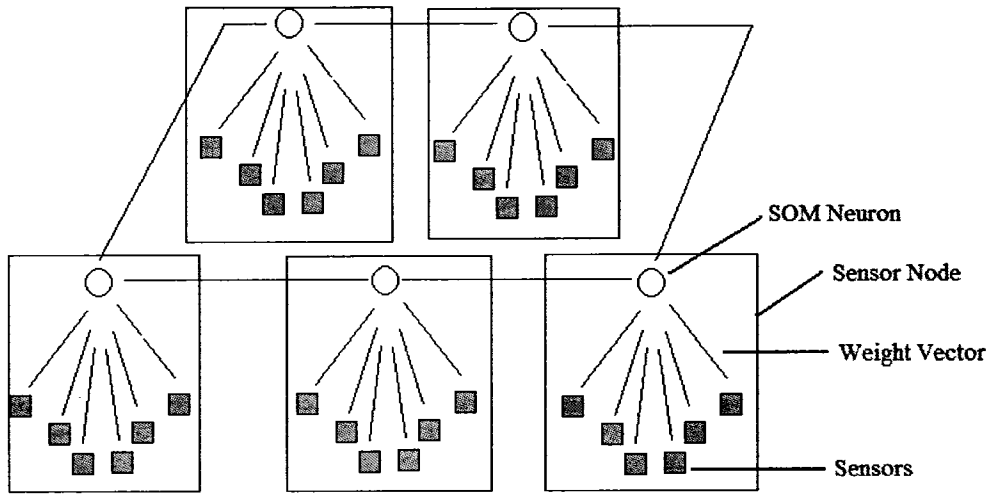

Figure 2.4: SOM model used in [30]

As shown in Figure 2.4, in the modified SOM model used by Caterall et al., each of the five neurons acts on a unique six-dimensional input vector, which corresponds to the six sensor readings that have been acquired locally by its six sensors. The assumption made by the authors is that the sensor readings will only vary slightly from node to node. As a result, each node's six-dimensional input vector will not be *exactly* the same, but will be very similar. Due to the SOM redesign employed, for every set of inputs, each node must only transmit one small data packet to the rest of the network. The packet includes an identification number belonging to the transmitting node, a timestamp, and Euclidean error between the node's weight and input vectors. Experimental results showed that after training, the SOM neurons organized to classify sensor patterns.

Zindel developed a thesis [31] in which a supervised-learning neural network implementation of a WSN was used to track the shadows cast by clouds as they moved across a field. Zindel's implementation used recursive subtrees of feed-forward, back-propagation neural networks. After training, the network achieved a high success rate of identifying cloud cover. The network's accuracy varied from about ninety percent to

about ninety-five percent depending on how many input/output pairs were used during the training phase.

Chapter 3

The WSN Problem

3.1 Problem Statement

WSN networking and processing software must achieve energy-efficient, fault-tolerant processing while simultaneously coping with node mobility and frequent node failures, and provide sufficient flexibility to allow for deployment in a wide variety of applications. For quick and easy deployment, detailed *a priori* knowledge of the target environment should be unnecessary. In other words, WSNs should adapt to their surroundings with minimal application-specific preprogramming and administrator oversight. To accommodate arbitrary numbers of nodes, architectures should be fully scaleable – the network should benefit from every additional node, and suffer equally from each node disconnect. The parallel processing capability of WSNs should be robust and versatile, suitable for deployment in a wide range of diverse applications. Since many sensed phenomena are represented by time-domain signals (such as voice audio, weather patterns, vibrations in machinery and so on), the WSN's processing algorithms should be sensitive to temporal patterns as well as spatial ones. With no *a priori* knowledge of the target environment, and therefore little knowledge of the input vectors that will be encountered, the WSNs must have the capability to recognize patterns and generate outputs based on input patterns of arbitrary length. For example, a military surveillance network may be designed to recognize human speech, though each word in

the target language is of different lengths, and therefore produce sequences with varying numbers of terms.

3.2 Analysis of State-of-the-Art Solutions

When evaluating the state-of-the-art of WSN protocols, it helps to have a historical context. While DARPA's DSN program was underway in the early 1980s, Dr. Robert Kahn was the director of the organization's Information Processing Techniques Office (IPTO). Dr. Kahn, a co-inventor of the TCP/IP protocol and a key figure in the development of the Internet [2, 32], was interested in applying the same approach of networking to sensor networks [2]. While this may have been a worthwhile endeavor two and half decades ago, the quantum leap that has meanwhile occurred in computer hardware technology allows for WSN networks whose performance requirements easily overwhelm traditional packet-switching protocols. Researchers have directed attention towards developing incremental changes to these protocols in the attempt to make them better suited to WSN applications, and perhaps through this evolutionary process, hereditary weaknesses are still evident in much of the current state-of-the-art.

Clustering algorithms in state-of-the-art architectures are intended to reorganize the network into multiple subgroups, each of which is of a more manageable size. There are, however, some drawbacks associated with the clustering algorithms employed by state-of-the-art WSN architectures. The organization of the network into clusters is an arduous task. For increased efficiency, clusters should be uniform in size and closely packed with minimal overlap. Meeting these targets requires either an iterative configuration process, or the making of unrealistic assumptions such as that every node

lies on a common two-dimensional plane (as in [25]), or that precise GPS information for every node is available (as in [21 & 25]). Once formed, clusters are fragile entities that are disintegrated by the death of the cluster-head, or as a result of their mobile constituents wandering too far apart. Due to their delicate nature, re-clustering is an ongoing process, with each repetition wasting time and energy. Therefore, clustering algorithms are not particularly resilient to node failures, are unable to effectively cope with node mobility, and require repeated energy expenditures in the form of overhead.

The most intrinsic quality of WSNs is their massive parallelism, which leads to a natural synergy with a parallel approach of processing their collected data. Since clustering itself does not lead to any inherent parallel processing capability, a second, application-layer protocol is needed to exploit the network's parallel hardware. Several state-of-the-art architectures apply artificial neural network (ANN) algorithms to achieve fault-tolerant parallel processing in sensor networks (for example, [28 – 31]). Neural networks share many inherent and desired characteristics with wireless sensor networks, such as relying on many simple local interactions to give rise to complex global behavior. Neural network implementations of WSNs are gaining popularity and attention among researchers [31], though several deficiencies are present in state-of-the-art neural WSN architectures. Supervised-learning ANNs (used in [31], for example) require *a priori* knowledge of the target environment, in the form of many input/output training pattern pairs. These input/output pairs must be crafted ahead of time by a human administrator (even if the pairs were generated by a separate computer system, that computer would also require *a priori* information, again necessitating some form of human involvement).

None of the state-of-the-art ANN-WSN architectures [38 – 31] are temporally sensitive. In other words, they can only perform a computational analysis based on the spatial distribution of input values and are indifferent to the time at which the inputs are occurring. As previously stated, these state-of-the-art protocols are incapable of recognizing patterns of events that are occurring in the time-domain.

Chapter 4

RSOM-WSN Architecture

4.1 Introduction

The architecture proposed in this thesis is referred to by the rather unimaginative name *RSOM-WSN*. RSOM-WSN is a neural-based architecture that is self-organizing, tolerant to both data errors and communication faults, and requires no *a priori* information about its environment. RSOM-WSN is also designed to recognize geographic and temporal signals. In other words, data patterns can be recognized regardless of their spacio-temporal locality. The underlying neural framework used is the Recurrent Self-Organizing Map (RSOM), which has never before been implemented in a sensor network setting. In RSOM-WSN, each sensor node acts as a neuron in an RSOM neural network, and therefore hardware sensor nodes act as complete analogues to RSOM neurons.

4.2 The Recurrent Self-Organizing Map (RSOM)

4.2.1 Overview

The recurrent self-organizing map is a powerful variant of Kohonen's original self-organizing map. First proposed by Varsta *et al.* in [34], the RSOM is a recurrent network that generates outputs based on a *sequence* of input vectors, rather than on a single input vector like the ordinary SOM.

4.2.2 RSOM algorithm in detail

4.2.2.1 RSOM input space

The RSOM is a dual-layer recurrent neural network, in which there is one input layer and one output layer. Input vectors originate at the input layer and are connected to each neuron in the output layer by a *weight* vector whose dimensionality is equal to that of the input vector. Each input vector sequence may be arbitrary in length, having N terms.

For example, suppose that a two-dimensional input sequence is as follows.

$$\textit{Dimension}\ \#1 \rightarrow (0.0,\ 0.5,\ 1.0,\ 1.5,\ 2.0,\ 2.5)$$
$$\textit{Dimension}\ \#2 \rightarrow (1.2,\ 4.5,\ 6.1,\ 7.9,\ 8.1,\ 9.5)$$
$$N = 6\ \textit{terms}$$

In this example, the input sequence consists of six terms, where each term is a two-dimensional input vector,

$$\textit{Input vector sequence} \rightarrow [0.0,\ 1.2],\ [0.5,\ 4.5],\ [1.0,\ 6.1],\ [1.5,\ 7.9],\ [2.0,\ 8.1],\ [2.5,\ 9.5]$$

4.2.2.2 RSOM neuron output function

Each neuron in an RSOM neural network generates its own output, determined by a recursive, iterative function that is re-evaluated as each input vector in the sequence is encountered. The output function is as follows.

$$y(n) = (1-\alpha)\,y(n-1) + \alpha\left(x(n) - w\right)$$

$$y(0) = 0$$

Where,

$y(n)$ is the neuron's cumulative output after the n_{th} term of the input vector sequence,

$x(n)$ is the n_{th} term of the input vector sequence,

w is the neuron's *weight* vector,

$0 \le \alpha \le 1$ is a learning constant

y, x, and w vectors with the same dimensionality

Equation 1. RSOM recursive output function

The output function can be considered as a *decay* of the previous level of output, $y(n)$, with a simultaneous *re-enforcement* of output caused by the most recent input term, $x(n)$. *Equation 1* can be re-written to explicitly show the contribution that each input term has on the cumulative output.

For a sequence with N terms, let $RSOM(x, w, \alpha) = y(N-1)$, *where*

$$y(N-1) = \alpha\left[x(N-1) - w + (1-\alpha)^1\left(x(N-2) - w\right) + (1-\alpha)^2\left(x(N-3) - w\right) + \ldots + (1-\alpha)^{N-1}\left(x(0) - w\right)\right]$$

Equation 2. RSOM output function (expanded)

Equation 2 explicitly shows the contribution each input term has on the neuron's cumulative output, and the function *RSOM(x, w, α)* is defined as the cumulative output after all N terms of the input sequence have been presented. Essentially, the RSOM output function computes an *exponentially weighted sum of biased inputs*. As shown in *Equation 2*, each input vector term, *x(n)* is biased by w, which is a neuron-specific modifiable parameter. Each biased input is then weighted such that the terms located towards the end of the sequence are given more weight than terms located early in the sequence.

4.2.2.3 RSOM neuron competition and the BMU

After every term in an input sequence has been presented to the network, each neuron computes an *absolute output*, |y|, which has a value equal to the magnitude of its output vector, *y*. Neurons "compete" by comparing their absolute outputs with that of the rest of the network. The neuron with the smallest absolute output wins the competition and is referred to as the *best matching unit*, or *BMU*. The BMU represents the neuron whose weights were most well trained with respect to the input sequence, that is, the neuron whose weights resulted in the least amount of output.

4.2.2.4 RSOM neuron weight vector and learning rule

Each RSOM neuron is associated with its own modifiable parameter known as a *weight vector*. The naming of this vector as a 'weight' by RSOM's inventors is to maintain some level of consistent terminology with other types of neural networks, and in this case is somewhat misleading. The weight vector in an RSOM neuron is not participatory in any multiplicative weighting, but rather serves as a *bias* or *offset*. Subsequently in this thesis, the weight vector may be referred to as an offset, or biasing term.

Initially, the weight vectors of every neuron are random. This represents a fully untrained state of the network, since no input sequences have yet been encountered. As input sequences are encountered, the weight vectors of neurons are modified according to a *learning rule* that is represented by the following *weight update formula*.

$$w_i(t+1) = w_i(t) + h_{ib}(t)\, y_i(t)$$

Where,
$w(t)$ is *the old weight vector,*
h_{ib} *is a neighborhood function,*
and y_i *is the neuron ' s output vector*

Equation 3. RSOM Weight-update formula

The neighborhood function in *Equation 3* is a function related to the distance between neuron *i* and the BMU. In typical learning schemes, $0 <= h <= 1$, and *h* produces its maximum value at the BMU, and decreasing values for every neuron at an increasing distance from the BMU's location. In other words, neurons near the BMU (including the BMU itself) will modify their weights by greater amounts than neurons that are farther away. The neighborhood function plays a key role in establishing the RSOM's topology-preserving characteristic.

A neuron's weight vector is modified according to the update formula in such a manner that if the same input sequence is encountered in the future, the corresponding output, *y*, will decrease. In this sense, a neuron's output, *y*, can be thought of as an *error* or *cost* function that the weight-update formula is attempting to minimize. Therefore, a neuron that has been fully trained to recognize a particular input sequence will produce an output of $y = 0$ whenever that input sequence is encountered. The *ideal* value of *w* such that $RSOM(x, w, \alpha) = 0$, can be calculated analytically as follows.

$$w = \frac{\sum_{k=0}^{N-1}(1-\alpha)^{N-1-K} x_k}{\sum_{k=0}^{N-1}(1-\alpha)^{k}}$$

where N is the number of terms in sequence x

Equation 4. Analytical computation of ideal weight for sequence *x*

4.2.3 Interpretation

The RSOM function computes an exponentially weighted sum of biased inputs. Neurons are trained to recognize common input sequences by finding a weight, *w*, which results in the RSOM function producing a zero output. For illustrative purposes, consider the following steps in computing an RSOM output (for the sake of simplicity, in the following example the input sequence is assumed to be one-dimensional). Also recall that the following computations are conducted by each neuron separately.

The first step of the RSOM neural algorithm is to offset the input sequence by the neuron's weight vector, *w*.

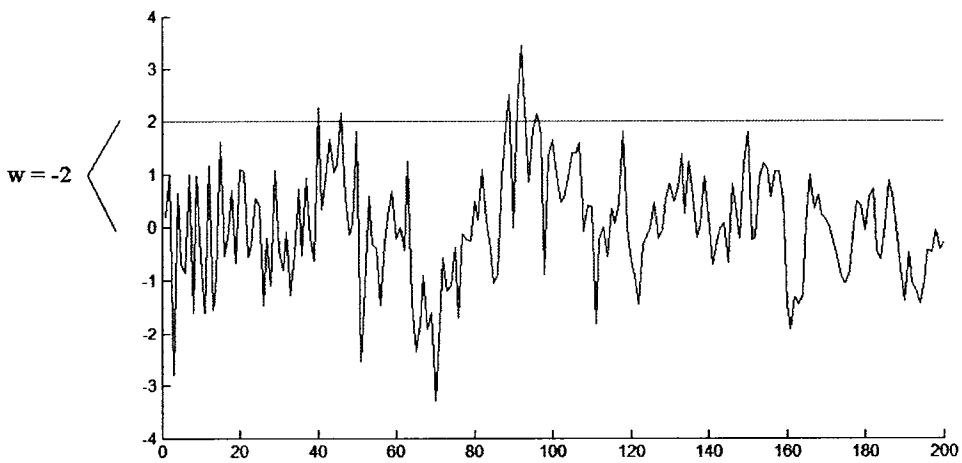

Figure 4.1: RSOM function step #1

The second step is to multiply the biased input by an exponential curve whose shape is determined by the parameter, α.

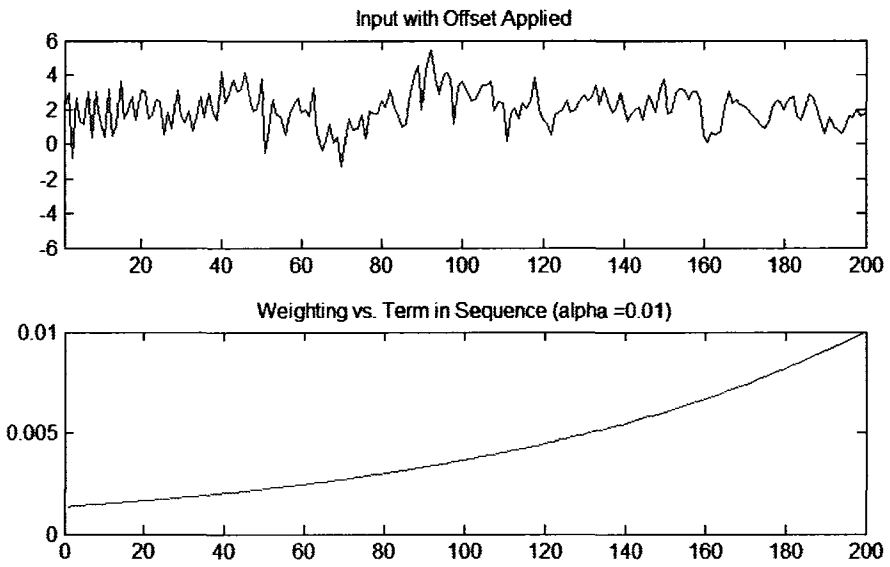

Figure 4.2: RSOM function step #2

The third step is to sum the value of each term. Graphically, this equivalent to finding the area under the curve produced at the end of step 2.

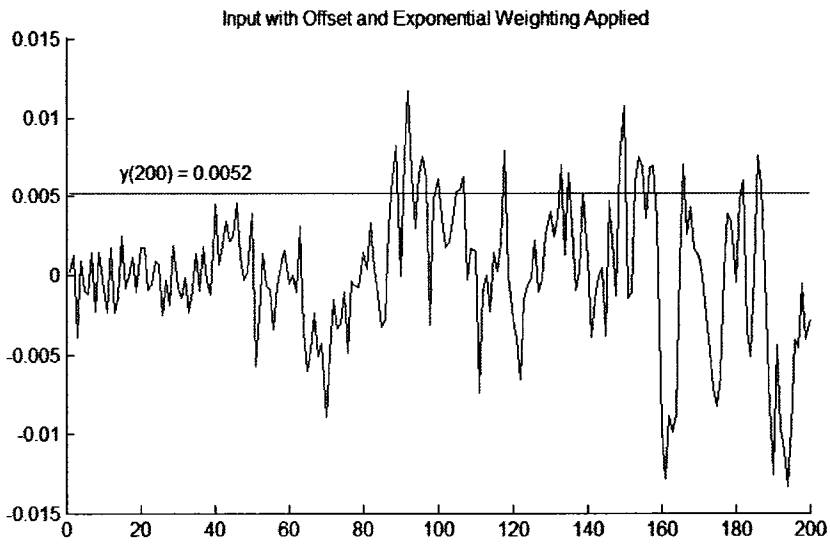

Figure 4.3: RSOM function step #3

In Figure 4.3, it is shown how terms near the end of the sequence have been amplified as a result of the exponential weighting performed in step 2. The RSOM algorithm results in a high degree of data compression, since a multi-dimensional sequence consisting of many terms is reduced to a single value, *y*.

4.3 RSOM-WSN Model

RSOM-WSN sensor nodes are directly analogous to RSOM neurons. Therefore, the number of neurons in the WSN-RSOM output layer is equal to the number of sensor nodes in the WSN-RSOM network. Additionally, since sensors are providing the neural inputs, and each node contains a sensor, every sensor in the network represents an element in the RSOM's input layer. Therefore, the dimensionality of the RSOM-WSN input vector (as well as all weight vectors, *w*, and output vectors, *y*) is also equal to the number of sensor nodes in the network (assuming each sensor node contains exactly one

sensor). Therefore, an RSOM-WSN network with fifty sensor nodes acts on a fifty-dimensional input vector, and contains fifty neurons in its output layer. The RSOM-WSN model is shown in Figure 4.4.

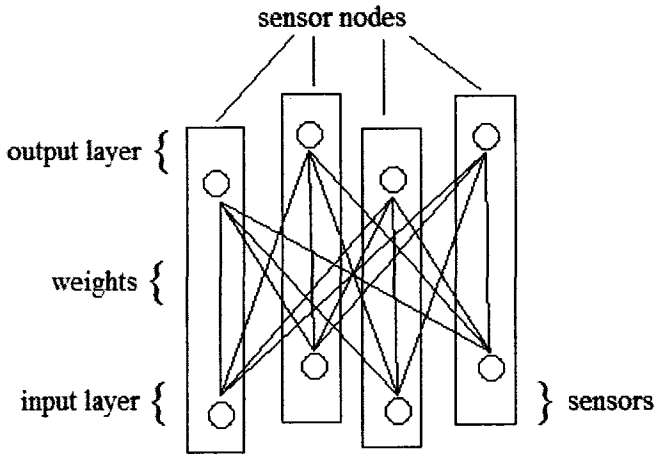

Figure 4.4: RSOM-WSN model with four sensor nodes

Typically, RSOM neural networks act on different input sequences that all have the same number of terms. The number of terms is known before hand, and is either by design or based on *a priori* knowledge of the input set. Wireless sensor networks, on the other hand, which act on input vectors that can be largely unknown at the time of deployment and also need the flexibility to operate in numerous applications, must have the capability to recognize patterns and generate outputs based on input patterns of arbitrary length. For example, a military surveillance network may be designed to recognize human speech, though each word in the target language is of different lengths, and therefore produce sequences with varying numbers of terms.

4.4 RSOM-WSN Protocols

The RSOM-WSN protocols proposed in this thesis serve as both networking and processing protocols. Whereas most wireless sensor networks implement networking and processing as separate layers (such as in TCP/IP), RSOM-WSN accomplishes both tasks simultaneously with dual-function packets.

An innovation of RSOM-WSN is in its data transmission strategy. Rather than using a *query-reply* strategy as seen in most state-of-the-art literature, RSOM-WSN employs what this thesis refers to as a *broadcast-react* strategy. Unlike *query-reply*, in which messages are directed to a specific destination, RSOM-WSN nodes merely broadcast messages, without any specific recipient in mind and without intending to respond to a query, or provoke a reply. The broadcasted messages flood the network, and will elicit neural reactions in any node that receives the signal. Sensor nodes employing the RSOM-WSN protocol are uncoordinated, largely unsynchronized, and completely autonomous. RSOM-WSN nodes are reactionary because their behavior arises in response to unscheduled stimuli. The protocol is a collection of event-driven processes that maintain node autonomy while also achieving global, intelligent, fault-tolerant pattern recognition. The wireless data packets that are broadcast from nodes are classified as one of three types, reflecting the purpose of the transmission. They are, *organizational, maintenance,* and *neural.* Organizational packets ensure that each sensor node possesses a unique node ID number, or address. Maintenance packets recalibrate neural weight vectors when node movement distorts them, and neural packets include all data necessary for input dissemination, weight updating, and ultimately pattern recognition.

4.4.1 Neural packets

There are three kinds of neural packets, *input, competition*, and *update*, numbered as packets 0 through 2, respectively. The broadcasting of any of these three packets from a node requires a triggering event, and broadcasted packets trigger a reaction in all nodes that receive them.

4.4.1.1 Input packets

Each node in RSOM-WSN performs a neural computation on a multi-dimensional input vector. The dimensionality of this vector is equal to the number of nodes in the network, and the value at each dimension corresponds to a sensor reading of a particular sensor node. Therefore, every sensor node must share its sensor data with every other node in the network. This data is transmitted in the input packet. Sensor nodes take regular sensor samples at a sampling frequency, F_S. After a node takes a sample, it broadcasts an input packet of the following form.

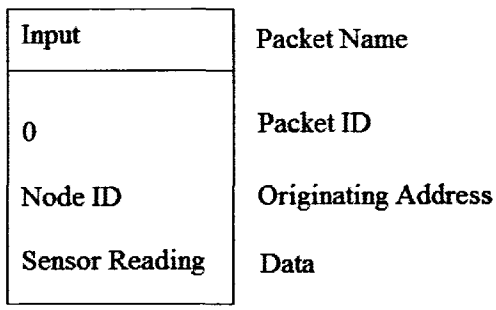

Figure 4.5: Input packet model

Every node will transmit an Input Packet every T_S seconds, where $T_S = 1/F_S$ is the sampling period. Therefore, any given node will receive a packet from every other node in the network during each T_S interval, and all of the packets received during this time are assembled to form a single input vector.

When a node receives an input packet, it checks to see if it has a weight associated with the Node ID that was included in the incoming packet. If it does not, then it creates a new weight, associates it with the Node ID contained in the packet, and initializes the new weight to a small random number. Subsequently, the node updates its own output with respect to the source of the packet. This characterizes the receiving node's reaction to the detection of the input packet. Pseudo-code for the reaction is shown below.

```
Input_Reaction {

        If weight(Packet_NodeID) does not exist {
            weight(Packet_NodeID) = small_random_number;
            y(Packet_NodeID) = 0;
        }

        y(Packet_NodeID) = (1 – α) * y(Packet_NodeID) + α * [NodeID_SensorReading
                                – weight(Packet_NodeID) ];
}
```

Notice that the node's output, y with respect to the broadcasting node is updated on an iterative basis, and is derived from *Equation 1* used in the RSOM algorithm.

4.4.1.2 Competition packets

Recall that each input sequence consists of N input vectors. Therefore, after N input vectors have been received and assembled by a node, it will begin competing with other nodes to determine which is the winner for the completed sequence. RSOM competition is based on the magnitude of the node's output vector y, so the node will compute $|y|$, and broadcast a competition packet structured as follows.

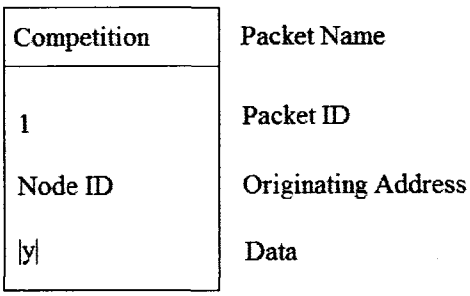

Figure 4.6: Competition packet model

When receiving a competition packet, nodes react by comparing the $|y|$ contained in the packet with their own $|y|$ that was computed locally. The winner of the competition is the node with the smallest output magnitude. In the case of two nodes having identical magnitudes of output, node ID number is used to break the tie. Pseudo-code for the reaction is shown below.

```
Competition_Reaction {

        If Packet_y < y {
                BMU = false;
        }
        if Packet_y == y {
                if Packet_NodeID < NodeID {
                        BMU = false;
                }
        }
}
```

In the above pseudo-code, *BMU* is a flag that is initially set as *true* as a default. When competition packets have been received from every node in the network, there will be one node whose internal *BMU* flag remains as *true*; this node is the winner, or BMU, for the sequence.

4.4.1.3 Update packets

After a node has established itself as a winner, it will broadcast an update message to neighboring nodes. This update packet is *not* transmitted at full strength, but rather at a lesser power level so that only nearby nodes will detect it. In effect, varying the power level of the update packet represents an implementation of the neighborhood function, h that is included in the RSOM algorithm's weight-update formula in *Equation 3*. Since all of the information in represented in the power of the signal, the packet only needs to include its identification as shown below.

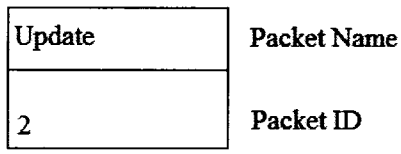

Figure 4.7: Update packet model

All nodes that receive an update packet, as well as the node that broadcasted it react by modify their weights according to the weight-update formula of *Equation 3*. When executing this formula, nodes calculate a value of h according to their distance from the winning node, which is inferred from the measured strength of the signal carrying the update packet.

4.4.2 Organizational packets

There is very little organizational overhead associated with activating an RSOM-WSN due to the geographically unaware, autonomous nature of its sensor nodes. All that must be established is the address, or node ID number of each sensor node. The node ID is unrelated to node location, capabilities, attributes, or any other software or hardware property. The only condition that must be satisfied is that each node has a unique identifier, so that the rest of the network can properly assign weighted connections to them. Essentially, unique identification numbers prevent logical or neural collisions from occurring in the sense that no two nodes will be sharing the same "virtual synapse". The assigning of node identification numbers in ROM-WSN is a straightforward process.

When nodes initialize during power-on, they each choose a random node ID for themselves. The range of possible numbers from which to randomly choose will depend on memory requirements and the number of nodes expected to be deployed in the

network. A reasonable estimate for a large-scale network is that the node ID number will be a two-byte value, allowing for 65,536 different node IDs to be simultaneously in use. During RSOM-WSN operation, each node broadcasts an input packet every T_S seconds. Therefore, if during a single T_S period, a node receives two or more input packets with the same encoded node ID number, this indicates that two or more nodes have randomly chosen the same identifiers, and neural collisions (or alternately, ID collisions) will be taking place. Once the receiving node has detected this neural collision, it broadcasts an organizational packet, which contains the node ID that has been overused.

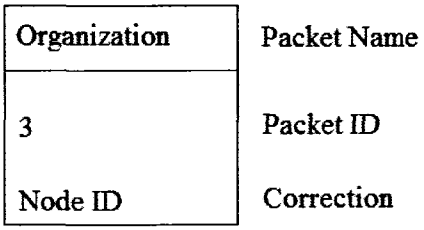

Figure 4.8: Organizational packet model

When a node receives an organizational packet, it compares its own node ID with that contained within the packet. If the same, it will recognize that another node has identified a neural collision, and the node will choose a new random identification number. It is possible that the fresh ID is also shared by another node, in which case a new organizational packet will inevitably be issued. After enough iterations, however, every node will possess a unique identifier. The number of iterations required will depend on number of nodes in the network compared to the number of unique IDs available (dictated by memory requirements). When the number of nodes is large compared to the number of available addresses, the required number of organizational packets can be unreasonably large. This is referred to as an *organizational stall,* since the network is

essentially stuck in the organizational phase, unable to progress to its functional operation. An organizational stall may be severe enough that it is more desirable to simply deactivate a number of nodes, thereby reducing the number of IDs that need to be allocated. Further, over a period of time, a large number of mobile nodes may migrate to a small area whose node density becomes such that there are more devices than available addresses to allocate to them. This can be thought of as two initially separate networks merging as a result of movement, and not enough IDs to service the aggregate network. This situation is referred to as an *organizationally unstable state,* since no number of iterations will be able to resolve the resulting neural collisions. To avoid this unstable state, as well as to eliminate severe organizational stalls, node deactivation is a possible reaction to the receipt of organizational packets. This *organizational deactivation* takes place when a node is involved in more neural collisions than some predetermined threshold. Deactivated nodes will occasionally reactivate, and attempt to rejoin the network by eavesdropping on network packets, and repeatedly selecting a node ID number until it finds one that is seemingly unused by other nodes. When a new ID number has been found, the node will be fully active and begin participating in network operations. In the case of a full or nearly full network (regarding the number of nodes vs. number of available addresses), deactivated nodes will inevitably take the place of nodes that have died as a result of damage or energy depletion.

4.4.3 Maintenance packets

Due to the presence of a geographically based neighborhood function, h, the weights of trained neurons are dependent on their relative locations. That is to say, neurons that are

located in a small neighborhood have similar weight vectors, and neurons that are far apart have a higher level of dissimilarity between their weight vectors. Therefore, weight vectors are partially a function of node location, and as node location changes as a result of mobility, their weight vectors must be subject to regular maintenance. The word 'maintenance' is being applied to this operation in order to clearly distinguish it from the 'updating' that occurs as a result of regular RSOM neural learning. The purpose of weight maintenance is to provide an estimated correction to the weight vectors of mobile nodes, so that the topology of the map does not become extremely distorted as a result of geographical reconfiguration. Every node broadcasts maintenance packets at regular intervals whose duration is related to sensor node density, and the expected speed of node movement. Nodes are intended to estimate the change in their physical location relative to other nodes by analyzing signal characteristics of the maintenance packets, such as measuring a change in signal strength over time (indicating a change in distance from the signal source), or through another mechanism such as the Doppler effect. A maintenance packet has the following form.

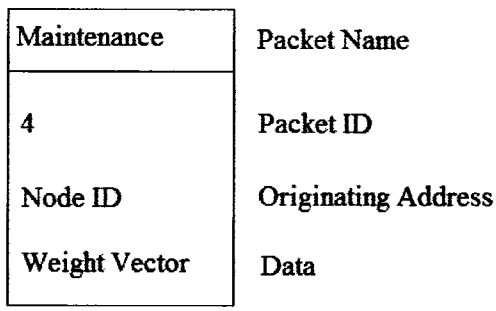

Figure 4.9: Maintenance packet model

When a node has received maintenance packets from the rest of the network, it calculates a correction for its own weight vector based on the weight vectors of other nodes, and its new position. The topology-preserving characteristic of self-organizing maps, including RSOM, leads to the formation of smooth gradients of weight vectors throughout the map.

The correction triggered by maintenance packets is intended to make the moving node's weight vector more similar to those of the nodes that it is approaching, to avoid topological spikes. Weight estimations can be calculated various ways, depending on required accuracy and support of computational complexity, though for the purposes of this thesis, weights are modified in the following way. By a receiving a maintenance packet, the node calculates the change in distance between itself and the packet's source. Based on the newly computed distance and the previous distance computed during the last round of maintenance, the node calculates a new weight for itself based on linear interpolation. For example, if a node finds itself moving 10% closer to a particular node, it computes an interpolated weight that is 10% more similar to the node that it is approaching.

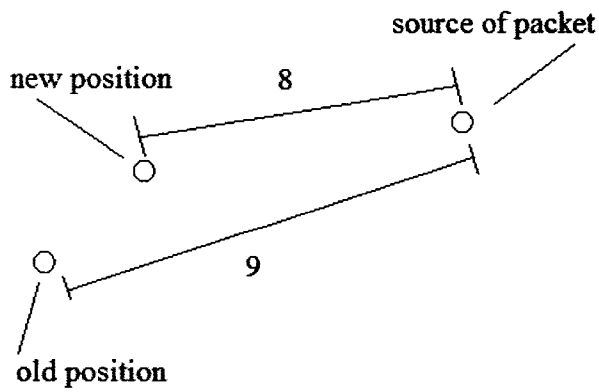

Figure 4.10: Movement of a sensor node

The scene in Figure 4.10 shows a mobile sensor node that has moved one unit closer to a stationary node. By analyzing the packet signal, the mobile node computes its new distance from the packet source, and recalls its old distance from that same source. The node then calculates the following adjustment.

$$weight_{adjust} = -\frac{D_{new}}{D_{old}}\left(weight_{packet} - weight\right)$$

where,
D_{new} *is the mobile node ' s new distance to the packet source*
D_{old} *is the mobile node ' s old distance to the packet source*
$weight_{packet}$ *is the weight vector included in the inbound packet*
weight is the node ' s own weight vector

Equation 5. Weight maintenance adjustment

The mobile node computes this linear interpolation for every broadcasting node within a certain neighborhood, and then adds the average interpolation to its own weight vector.

Chapter 5

RSOM-WSN Analysis

The RSOM neural model must be investigated analytically in evaluating its potential as a pattern recognizer in real-world sensor network environments. The extension of the feed-forward self-organizing map to a recurrent version clearly facilitates a degree of temporal-sensitivity to input sequences, though the sequences addressed in prior RSOM literature were usually small, consisting of just a few terms. The selection of parameter values such as α and h can be done by inspection for these small sequences, or alternatively by trial and error until an acceptable result is achieved. On the other hand, prediction of an RSOM's behavior when presented with very large sequences (containing say, ten thousand terms) is not as easy to inspect or test through trial and error. A comprehensive mathematical analysis is therefore needed to develop an understanding of how RSOM will react to real-world input sequences, such as would exist in a sensor network application

5.1 Time-Domain Analysis

An important characteristic of the RSOM function is exposed when its input is a sequence of constant terms, as follows.

$$RSOM(x, w, \alpha) = \alpha\left[x - w + (1-\alpha)^1(x-w) + (1-\alpha)^2(x-w) + \ldots + (1-\alpha)^{N-1}(x-w)\right]$$
where $x(0) = x(1) = \ldots = x(N) = x$ *is a constant* $\cdot$

Equation 6. RSOM with constant input

Recall that the weight vector is also constant in each term of a single sequence, and is only modified by the learning rule between successive sequences. Therefore, *Equation 6* can be rewritten as the following geometric series.

$$RSOM(x, w, \alpha) = \alpha \cdot x' \sum_{k=0}^{N-1} r^k$$
where $x' = (x - w)$ *is a constant biased input,*
and $r = (1 - \alpha)$

Equation 7. RSOM geometric series

As shown in *Equation 7*, the RSOM function is equivalent to a geometric series when the input terms are constant. Therefore, where the number of terms in the sequence, *N*, is very large, the final value RSOM output resulting from constant inputs is computed as follows.

For sufficiently large N,
$$RSOM(x, w, \alpha) = \alpha\left[\frac{x'}{1-r}\right] = \alpha\,\frac{x'}{1-(1-\alpha)} = \frac{\alpha}{\alpha}\,x' = x'$$
where $x' = (x - w)$ *is a constant biased input,*
and $r = (1 - \alpha)$

Equation 8. RSOM infinite geometric series

Equation 8 shows that when input terms are constant, the RSOM output will converge to the value of the biased input terms, $x' = (x\text{-}w)$, regardless of the value of α.

This parameter, will, however, affect the rate of convergence. The step response of the RSOM function is shown below for different values of α.

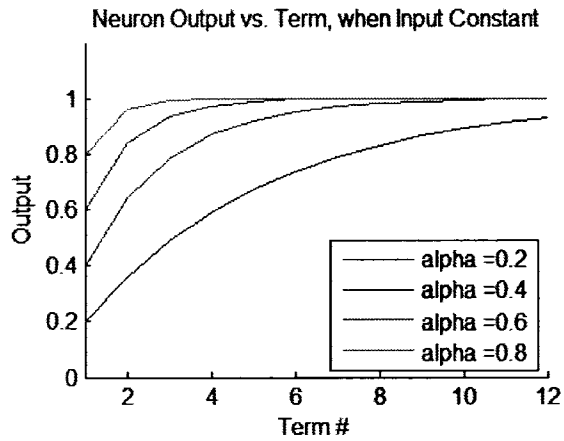

Figure 5.1: RSOM step response vs. α

The rise-time of the RSOM step response can be analytically derived as the following function of α,

$$t_r = T_S\left\lceil -\frac{1}{\log_{10}(1-\alpha)} - 1\right\rceil - T_S\left\lceil \log_{10}\frac{0.9}{\log_{10}(1-\alpha)} - 1\right\rceil$$

Where,
T_S *is the sampling period,*
and $\lceil\ \rceil$ *is the ceiling operator*

Equation 9. RSOM step-response rise time

As shown in *Equation 6,* neural output decays exponentially by being multiplied by increasing powers of (1-α), but is never totally eliminated. Therefore, RSOM neural activity acts as an infinite-impulse response (IIR) digital filter, meaning that in reaction to an impulse input, the output will rise and never decay all the way to zero. Every input results in a lingering output that is always present thereafter. The time-domain impulse response of neural output is shown in Figure 5.2.

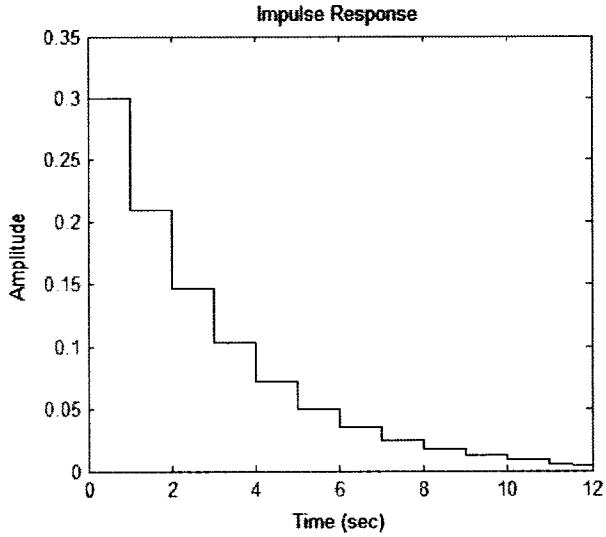

Figure 5.2: RSOM impulse response ($T_S = 1.0$, $\alpha = 0.3$)

5.2 Frequency-Domain Analysis

RSOM neural output is a discrete-time LTI (linear, time-invariant) function, and therefore can be represented by a corresponding z-domain transfer function.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\alpha}{1 + (\alpha - 1) z^{-1}}$$

Equation 10. RSOM transfer function, z-domain

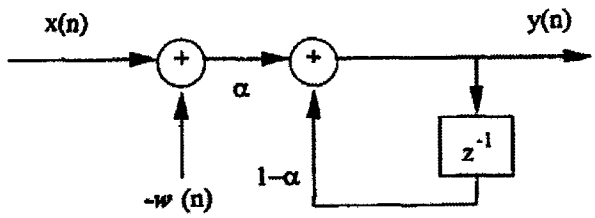

Figure 5.3: Schematic picture of an RSOM filter (image taken from [35])

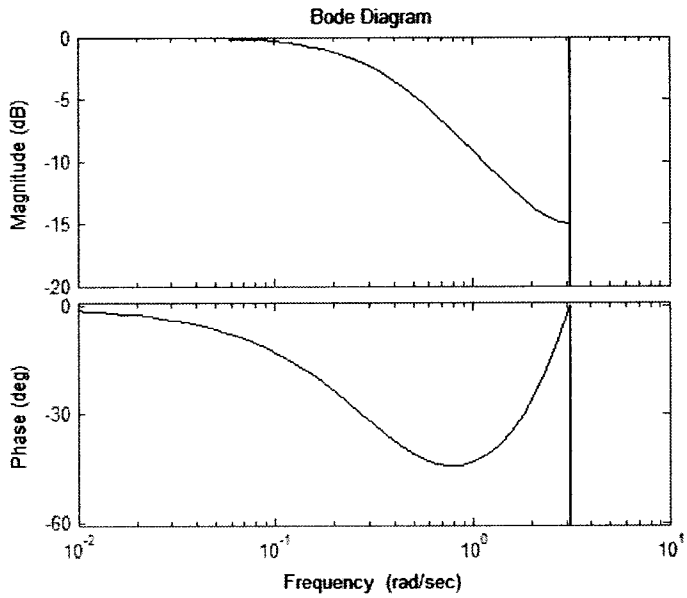

Figure 5.4: RSOM frequency response ***(TS = 1, α = 0.3)***

The frequency response of the system in *Equation 10* is shown in Figure 5.4. The frequency response illustrates RSOM's behavior as a low-pass filter, with increasing attenuation occurring at all frequencies above a particular threshold. This cut-off frequency, ω_C, above which signals are attenuated, can be calculated analytically.

$$\omega_C = \frac{2}{T_S} \tan^{-1}\left(\frac{\alpha}{2-\alpha}\right) rad/s$$

Equation 11. RSOM cut-off frequency

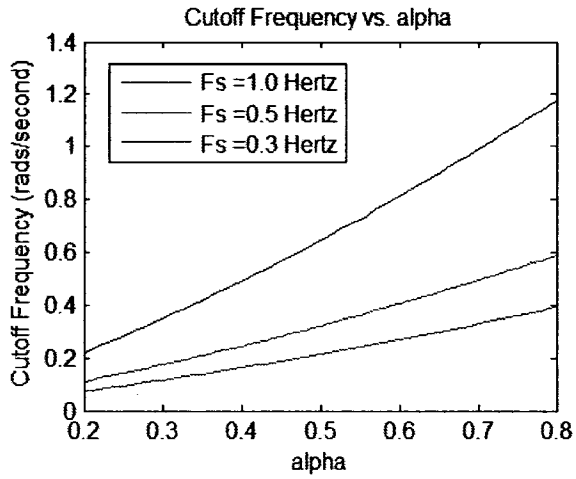

Figure 5.5: Cutoff frequency vs. α

A plot of cut-off frequency against α and sampling frequency F_S is shown in Figure 5.5. The RSOM low-pass filter allows higher frequencies to pass through without attenuation as α increases. Increasing sampling frequency, $F_S = 1/T_S$, also increase ω_C.

5.3 Configurable Parameter - Alpha

The parameter, α, plays multiple roles during RSOM neural operation. It can be referred to as an activation parameter, a decay parameter, or a memory parameter. A constant, it must be tuned to the target application prior to network deployment, though to achieve neural stability, the value of α is always between zero and one.

$$0 \leq \alpha \leq 1$$

Equation 12. Range of α

Due to the exponential decay of neural activation, terms towards the end of an input sequence always provide a greater relative contribution to total neural activation than terms located early in the sequence. The relative weighting by which each input term contributes to the total neural output can be calculated as a function of α, the

position of the term in the sequence, and the total number of terms in the sequence, as in *Equation 13.* A plot of term weightings vs. α for a sequence with ten terms is shown in Figure 5.6.

The relative weight of the nth term of a sequence with N terms,

$$weighting = \frac{1}{\sum_{k=0}^{N-1}(1-\alpha)^{k+n-N}}$$

Equation 13. Relative term weighting

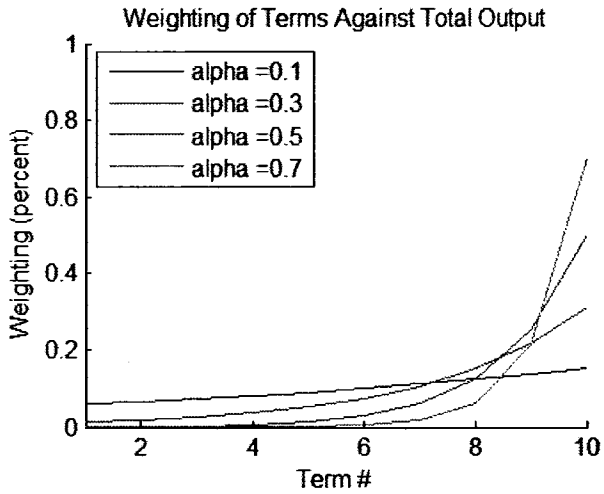

Figure 5.6: Term weightings vs. α for *N=10*

The weightings calculated in *Equation 13* are expressed as a fraction of total neural activation, such that the sum of the weightings of all terms is equal to 1, as is the area under each of the curves in Figure 5.6. Small values of α result in a slow decay of old input terms, therefore the weighting of terms is more uniform, as shown in Figure 5.6 (*α=0.1*). Networks whose activations decay slowly are said to have long-term memory, because their output behavior is significantly influenced by old input terms. On the other hand, large values of α (such as *α=0.7* in Figure 5.6) result in a rapid decay of neural activation, therefore when the end of a sequence is reached, little activation caused by the

first few terms remains. Networks of this type are said to have short-term memory, since only the most recent terms have any meaningful affect on neural output. At one extreme, $\alpha=1$, the neural output is based entirely on the most recently presented input term, and the RSOM acts exactly like Kohonen's original self-organizing map.

5.4 RSOM Identities

For a better understanding of RSOM behavior, several identities can be derived from the first principle of neural activation in *Equation 2.* In the following identities, sequences x and y both are composed of an equal number of terms, and the omitted value of α is constant.

$$RSOM(x+y, w) = RSOM(x, w) + RSOM(y, 0) = RSOM(x, w) + RSOM(y+w, w)$$

Equation 14. RSOM identity #1

$$RSOM(x, w_1) + RSOM(y, w_2) = RSOM(x+y, w_1+w_2)$$

Equation 15. RSOM identity #2

$$\begin{aligned} &\text{if } RSOM(x, w_x) = 0 \;\&\; RSOM(y, w_y) = 0, \\ &\text{then,} \\ &RSOM(x+y, w_x+w_y) = 0 \end{aligned}$$

Equation 16. RSOM identity #3

$$\begin{aligned} &\text{if } RSOM(x, w_x) = 0 \;\&\; RSOM(y, w_y) = 0, \\ &\text{then,} \\ &|RSOM(x, w_y)| = |RSOM(y, w_x)| = |RSOM(x, 0) - RSOM(y, 0)| \end{aligned}$$

Equation 17. RSOM identity #4

Chapter 6

Simulations

Simulations of RSOM neural algorithms have been conducted by a number of researchers, leading to observations appearing in several publications (such as [34, 35 & 36]). These simulations have illustrated RSOM's ability to adapt to arbitrary input patterns, recognize epileptic activity present in electroencephalogram (EEG) signals, and so on. The fundamental RSOM neural algorithms for activation levels (*Equation 1*) and the learning rule (*Equation 3*) remain unchanged in the WSN architecture proposed in this thesis. Therefore instead of "re-simulating the wheel" as one might say, more important simulations involve behaviors that arise specifically from fusion of the RSOM neural network with wireless sensor network hardware. This includes all WSN-specific phenomena such as packet loss, mobility, sensor node hardware limitations, etc. The simulation engine used for RSOM-WSN investigations was purpose-built as a part of this thesis. Coded in *C++*, the object-oriented software simulated a virtual network of sensor nodes, not simply neural units. In other words, the simulations included the lack of functionality that results from each sensor node being an autonomous entity. The simulator records all communication between nodes, and nodes can only communicate by generating an intentional packet. The *C++* simulator maintains detailed log files, which are exported to MATLAB for analysis.

6.1 Learning Simulation

The first simulations conducted were those involving input sequences far longer than anything encountered in previous RSOM literature. In this experiment, a simple RSOM-WSN network of two neurons were exposed to two different input sequences, referred to as "signal A", and "signal B", as shown in the following figures.

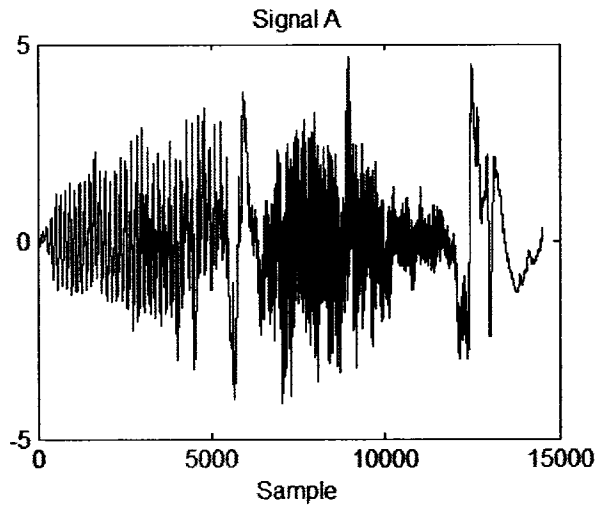

Figure 6.1: Signal *A*

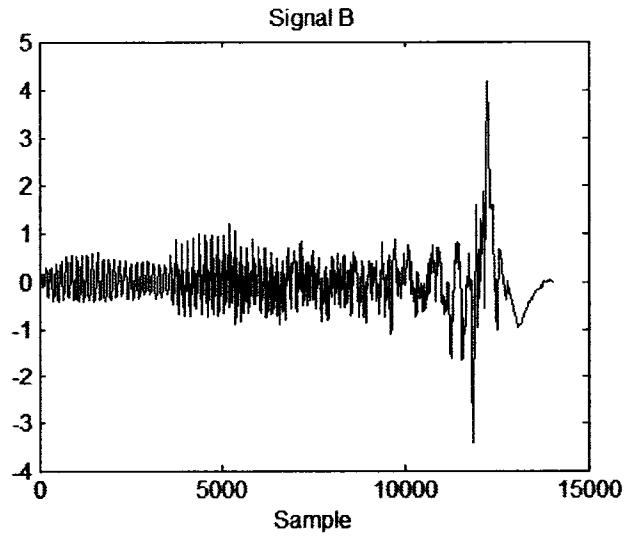

Figure 6.2: Signal *B*

During training, signal A was exposed to the network one hundred times, followed by an equal number of signal B's. The levels of activation for each of the two neurons can be seen in Figure 6.3 and Figure 6.4, along with their weight vectors.

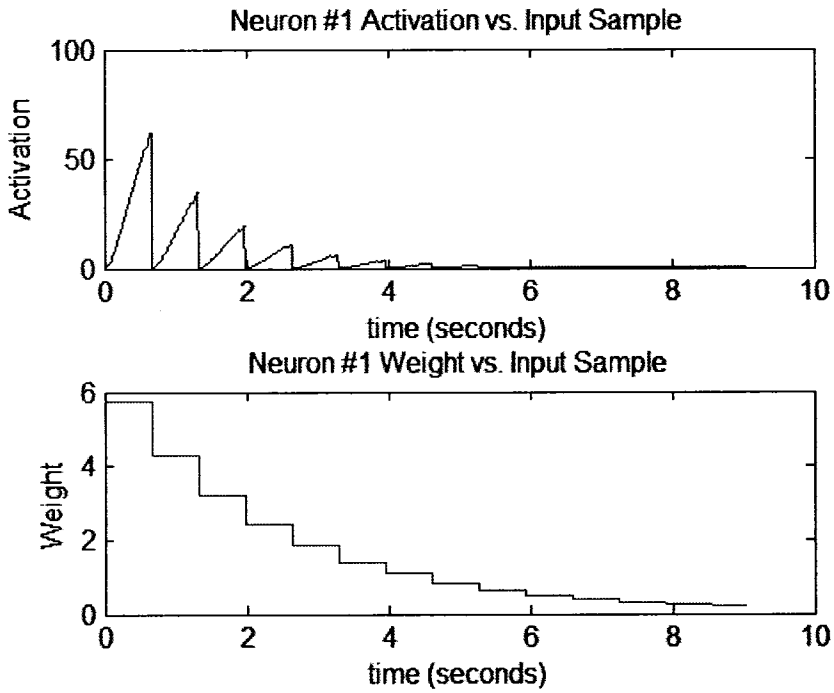

Figure 6.3: Neuron #1 activation (first 14 epochs)

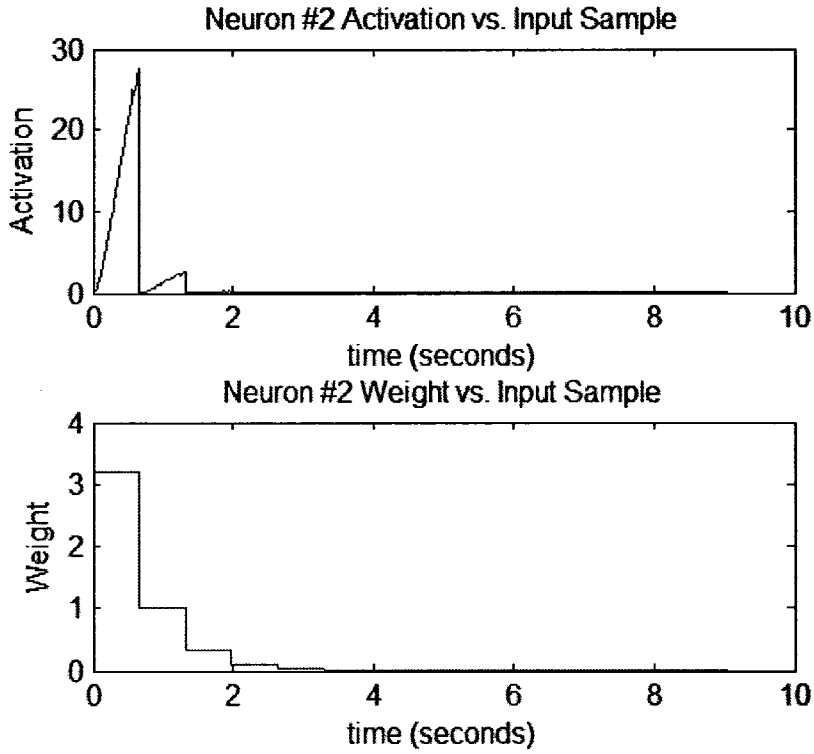

Figure 6.4: Neuron #2 activation (first 14 epochs)

As shown in Figure 6.3 and Figure 6.4, neuron #2 was the BMU (best matching unit) for signal A, and therefore its weights were adjusted much more than that of neuron #1. However, after signal B had been presented to the network, neuron #1 quickly became the BMU and hence associated with that signal. After the initial 200 rounds of training, the neurons were able to recall their respective signals even when a great deal of noise was added. The simulation was conducted many times with different values of α, and also with varying amounts of noise in the input signals. Figure 6.5 and Figure 6.6 show the total output of each neuron when presented with noisy versions of signal B. The signal B had been corrupted with additive white Gaussian noise, whose severity is measured by signal-to-noise ratio (SNR). Since neuron #1 had been associated with

signal B through learning, it should ideally be capable of recognizing the noisy signal as well, and therefore have less activation than its rival, neuron #2.

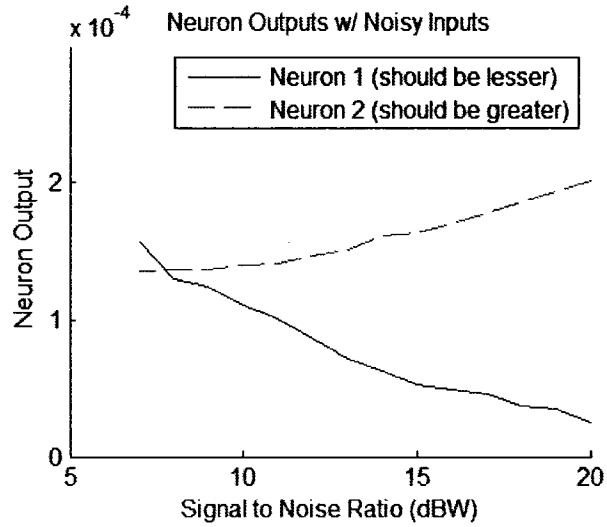

Figure 6.5: Neural response to noisy signal B (α = 1e-6)

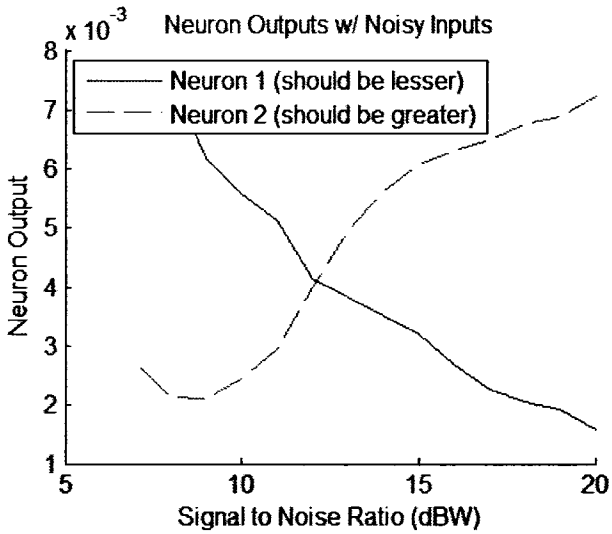

Figure 6.6: Neural response to noisy signal B (α = 1e-6)

As seen in the figures above, the neuron was able to recall its learned signal despite significant noise corruption. Also noteworthy is that the effect of the noise tends to decrease with the value of α.

6.2 Organization

The only network initialization needed in RSOM-WSN is the selection of a unique identification number by each node. This initialization phase involves an iterative process of repeated re-selection of random identifiers. When two nodes share the same ID number, a neural collision is said to occur, and those nodes participate in the next iteration of random selection of new identifiers. Simulations were conducted to investigate the number of iterations necessary to eliminate all neural collisions, and show the decrease in collisions achieved through each iteration. In each simulation, node ID numbers were associated with a one-byte memory; 256 unique identifiers were available.

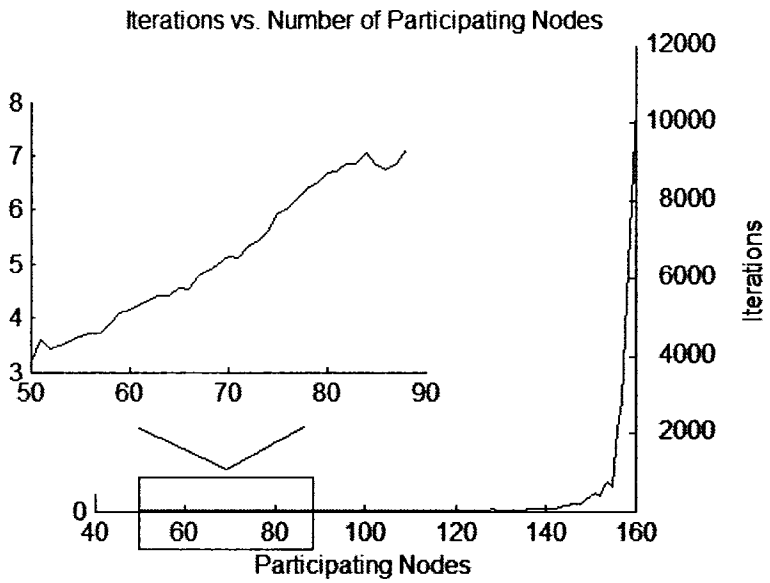

Figure 6.7: Total number of iterations vs. node population (*inc. subplot magnification*)

Figure 6.7 shows the total number of iterations that were required to eliminate all neural collisions is a newly established RSOM-WSN network, when organizational

deactivation was not employed. When the number of participating nodes is small compared to the number of available identifiers, the number of required iterations is small, and increases linearly with the number of nodes. When the number of nodes increased to about seventy percent of the number of available identifiers, there occurred an exponential increase in the number of required iterations. For example, when 160 nodes were sharing the 256 available node ID numbers, 10,000 iterations were required to allocate unique IDs to each node. Figure 6.8 shows the number of neural collisions that took place, per iteration, with 150 nodes in the network.

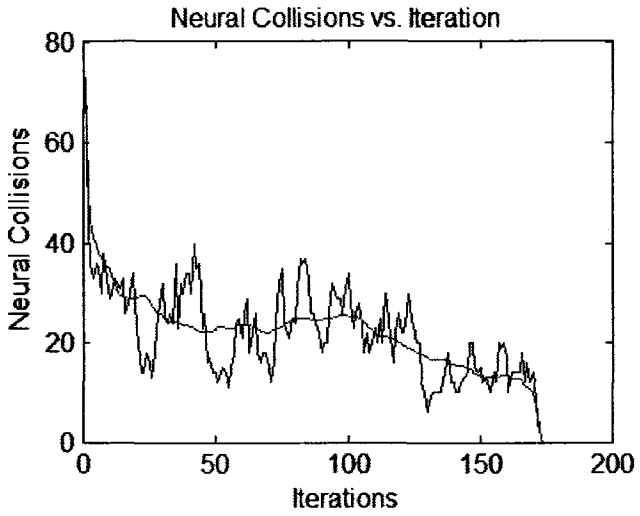

Figure 6.8: Number of neural collisions vs. iteration (150 nodes; incl. raw & smoothed)

As is the trend with the organizational protocol of RSOM-WSN, there tends to be a rapid decrease in the number of collisions during the first few iterations, and then an oscillation in which collision rates increase and decrease for many further iterations. As shown in the figure, approximately 175 iterations were necessary to reduce the number of neural collisions to zero. Figure 6.9 shows a similar graph, when 160 nodes were present in the network.

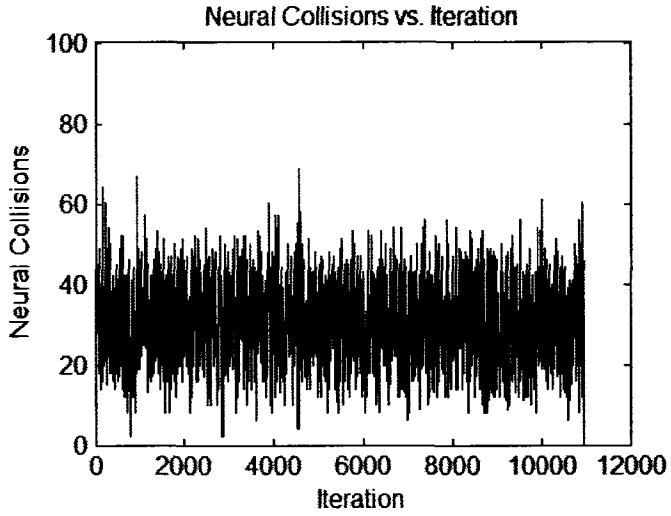

Figure 6.9: Neural collisions vs. iteration (160 nodes)

As shown in Figure 6.9, approximately 11,000 iterations were necessary to fully organize the network identifiers, when organizational deactivation was not employed. When organizational deactivation was implemented in the same scenario, a drastic decrease in the number of required iterations was observed.

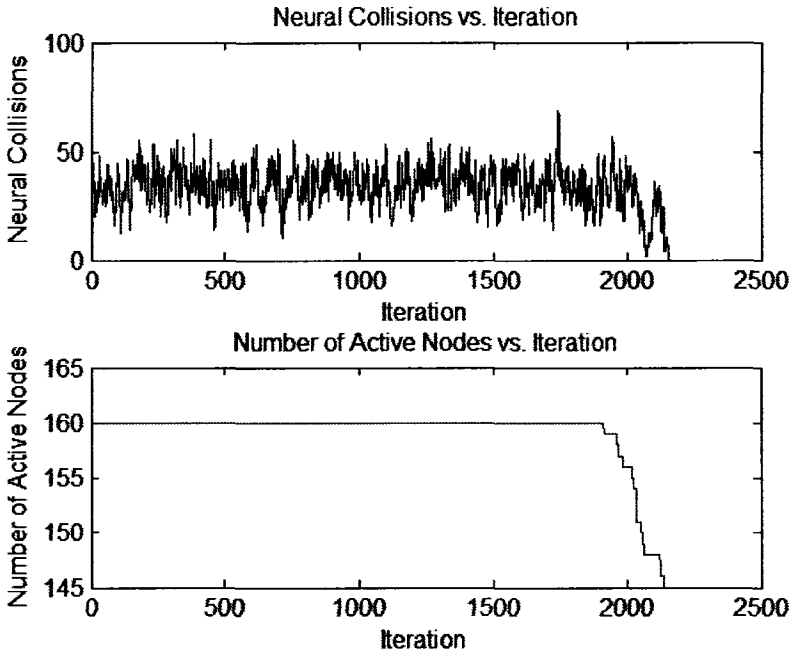

Figure 6.10: (top) Neural collisions vs. iteration w/ deactivation (160 nodes) (bottom) Number of remaining active nodes vs. iteration

Figure 6.10 shows the number of neural collisions per iteration when organizational deactivation is employed. In this simulation, each node was programmed to deactivate whenever it become involved with more than 500 neural collisions during its lifetime. This lead to an 80% reduction in the number of organizational iterations, and only necessitated the deactivation of 15 nodes, or 9.4% of the total node population.

6.3 Mobility / Weight Maintenance

The weight vectors of mobile nodes must be continually modified in an incremental manner to maintain the topology-preserving characteristic of the RSOM neurons. The following simulation was designed to test the efficiency of the weight-adjustment algorithm described in *Equation 5* and more generally in section 4.3.3. The simulation was based on a fully trained RSOM neural network as presented in [34]. This network had been trained to recognize the following input pairs: (1,1), (1,3), (1,5), (3,1), (3,3), (3,5), (5,1), (5,3), (5,5). If the pairs are labeled 1 though 9, such that (1,1) = 1; (1,3) = 2; ... (5,5) = 9, then there exists 81 patterns *XY* in which *X* and *Y* each represent an input pair, and can be referred to by their corresponding symbol. For example, the pattern *XY* = *12* = *(1,1), (1,3)*. The fully trained RSOM in [34] has learned to recognize each of the 81 patterns, and self-organized as shown in Figure 6.11.

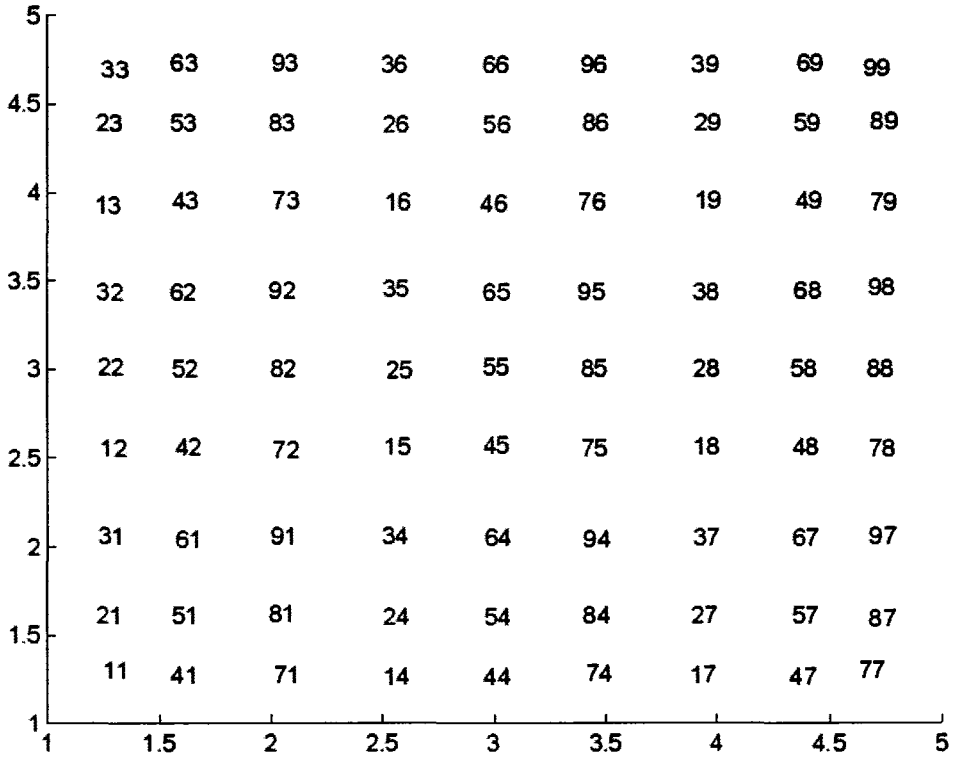

Figure 6.11: A trained RSOM (taken from [34])

In Figure 6.11, 81 neurons are shown according to their geographic location, each of which has become associated with a particular input pattern. Given a value of α and *Equation 4*, the two-dimensional weight vector for each neuron was calculated, and the topological mapping of weight vectors is shown in Figure 6.12.

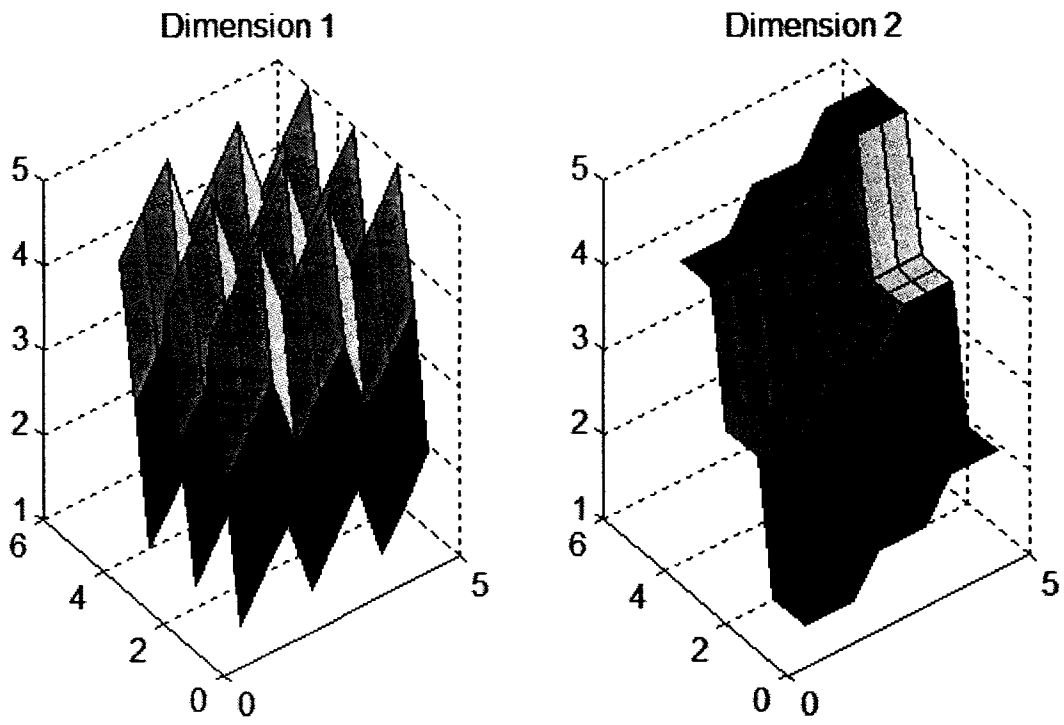

Figure 6.12: Two-dimensional weight-vector gradient vs. topology
(left) Dimension 1; (right) Dimension 2

To test the weight-update algorithm, simulations were conducted in which a node from coordinate (2, 1.5) traveled along a path and finally arrived at coordinate (3.5, 4.5). Firstly, the node made the journey without updating its weights. The resulting topology of weights is shown below.

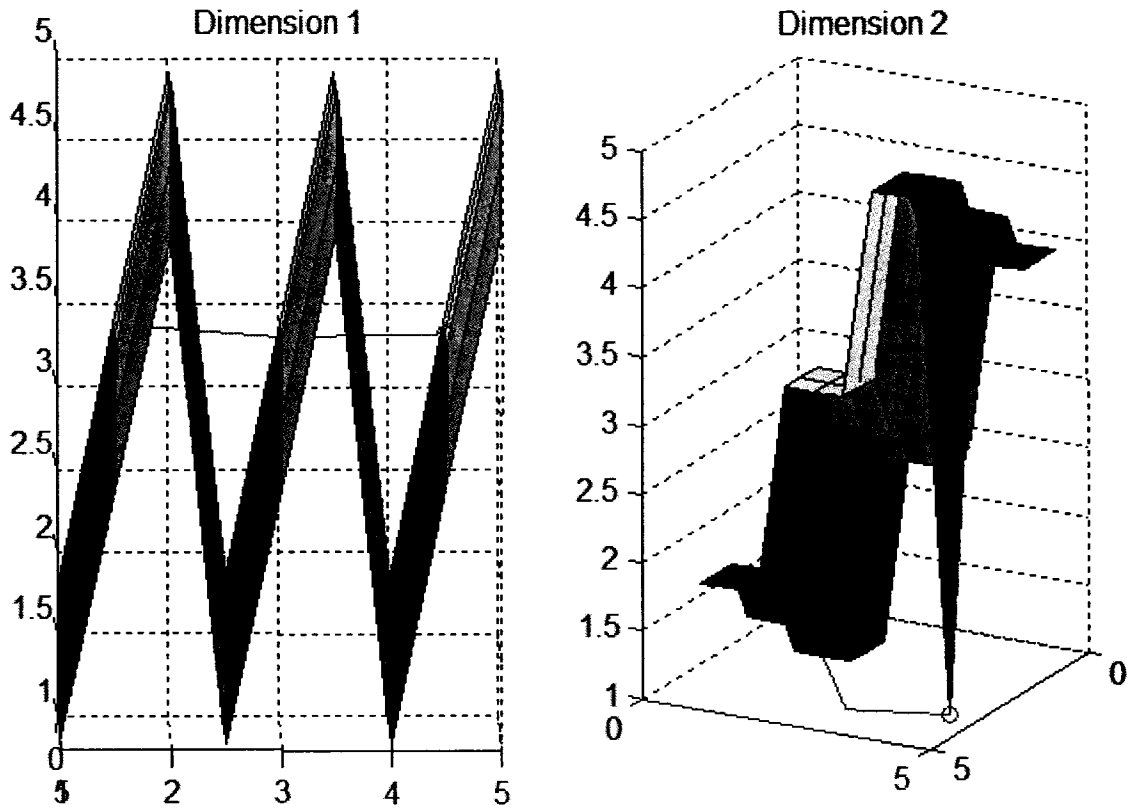

Figure 6.13: Weight topology after node movement (no adjustment)

In Figure 6.13, the new weight topology is shown after a node has moved along a path (marked with a line) to a new position without adjusting its weight vector. Notice that the first dimension of the weight vector is not distorted, because the original weight vector of the mobile node was identical to the ideal value at its destination. However, had the node stopped its journey any earlier, a distortion would have resulted as can be seen from the path marked in red. In the second dimension, however, a very large distortion in the weight topology has occurred. If the network would continue operating in this state, the spike in the 2nd dimension would eventually become smaller and smaller through the normal weight-updating process in *Equation 3*, but the accuracy of network outputs will be severely reduced until the smoothing occurs (possible taking several

hundred or thousand learning epochs). A simulation of the same mobile node along the same path was conducted again, but with the weight-maintenance adjustment implemented as described in section 4.3.3.

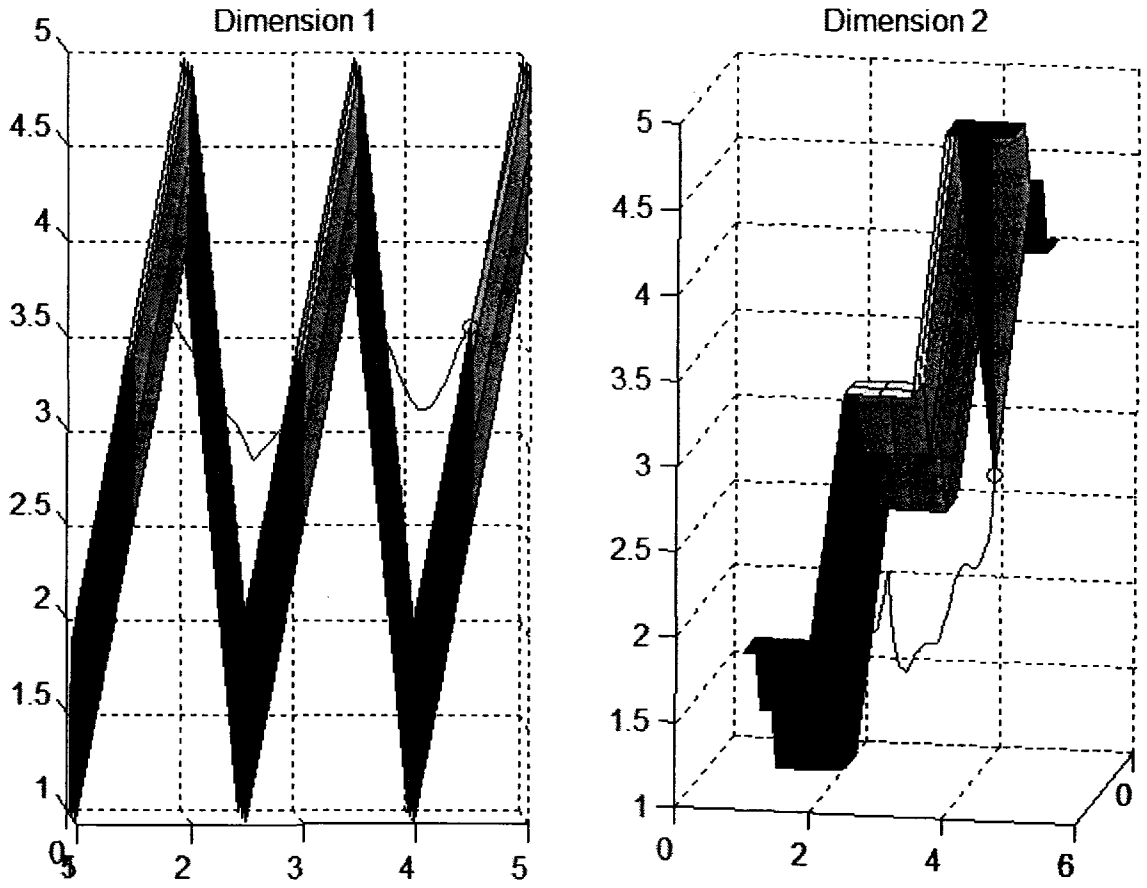

Figure 6.14: Weight topology after node movement (with adjustment)

Figure 6.14 shows the improvement made by the weight-maintenance adjustments. The blue path shows the value of the weight vector of the mobile node as it moves along its path. In the 2nd dimension of the weight gradient, the spike occurring at the destination coordinate is far less severe than in the unadjusted case shown in Figure 6.13. Also, in the first dimension, the figure shows how the weight vector of the mobile node tends to adjust itself well as it moves along its path, as indicated by the blue line

'following' the trends in weight vectors of nearby nodes. The result is that the network will smooth itself out in much less time than in the unadjusted case, therefore providing accurate outputs over a greater length of time.

Chapter 7

Summary and Conclusions

7.1 Summary

The work detailed in this thesis describes a new architecture for wireless sensor networks called RSOM-WSN. The proposed architecture is parallel, adaptive, self-organizing, unsupervised, and resilient against noise and other faults. The design of RSOM-WSN was inspired by the intelligent behavior of artificial neural networks, whose inherent and desired characteristics closely resembled those of wireless sensor networks.

7.2 WSN Problem Statement

Analysis of wireless sensor network literature revealed that state-of-the-art WSN architectures lacked synergy with their corresponding sensor node hardware. Existing architectures often required sizeable overhead during configuration, and were not equipped to deal with phenomena such as frequent node death, node mobility, and so on. State-of-the-art architectures also seemed to view the configuration and maintenance of the network to be separate tasks from the parallel processing of input data. This thesis contends that networking and processing are best achieved using multi-purpose recursive algorithms. These algorithms must be flexible to succeed in a multitude of applications, require little configuration overhead, be resilient against sensor node death and input noise, and support node mobility.

7.3 Theory and Simulation

The RSOM-WSN architecture proposed in this thesis is based on the recurrent self-organizing map, which is an uncommon neural network designed to process and recognize multi-dimensional temporal sequences. Mathematical analysis of the RSOM neural algorithm was necessary to establish a solid framework for comparison to state-of-the-art algorithms as well as to understand the applications and environments in which the proposed architecture would succeed. Due to the massively parallel nature of wireless sensor networks, simulations were also necessary to investigate emergent behaviors that were too complex to address analytically. To conduct the simulations in question, this author developed a purpose-built, object-oriented simulator that was highly configurable and designed to investigate several types of network behavior. Simulations investigated network configuration, node mobility, neural activations and resilience against signal corruption caused by noise.

7.4 Results

Simulation results revealed the network's ability to quickly self-organize an arbitrary number of participating sensor nodes. The organization required very little overhead, and was cooperatively achieved by nodes without the command and control of an external system or internal cluster-head. Once deployed, neural learning took place as expected, and the learning of several large temporal sequences was simulated. The effect of noise on the network's accuracy depends on the specific sequences that have been learned, and the type of noise encountered, though in simulations conducted as part of this thesis, accurate pattern recognition commonly took place with signal-to-noise ratios of less than

10dBW. Weight vector maintenance adjustments significantly decreased the topological disturbances resulting from node movement. Movement – especially that which is fast compared to the network's sampling rate – invariably results in some degree of weight vector gradient distortion, though this represents, at worst, a temporary loss of accuracy rather than a permanent one, since distortions are eventually eliminated through the network's normal weight-updating rule included in neural learning.

7.5 Contributions

The specific contributions of the work in this thesis are the RSOM-WSN architecture, and a more detailed mathematical analysis of RSOM neural behavior than has been presented in any discovered literature. The RSOM-WSN architecture is novel compared to state-of-the-art solutions in several ways. In RSOM-WSN, each sensor node imitates an RSOM neuron, that is to say, RSOM-WSN sensor nodes and RSOM neural units are complete analogues. Also, RSOM-WSN approaches networking and parallel processing as a combined task, accomplished with combined algorithms, rather than as separate objectives implemented using several layers of different algorithms as seen in other state-of-the-art research.

7.6 Recommendations for Future Work

The RSOM-WSN architecture, while sufficiently capable of meeting the design objectives described in this thesis, is still limited in comparison to the extensive range of applications for which the deployment of sensor networks has caught the interest of researchers. The most significant limitation of RSOM-WSN is its geographical range.

The proposed architecture requires every node to be within communication range, therefore as the distance between the furthest-separated nodes increase, an exponential increase in communication power is required to form the network. In the future, the neural algorithms of RSOM-WSN should be modified to address this issue. The theoretical solution would differ functionally from the proposed RSOM-WSM in the following way. Whereas in the proposed RSOM-WSN, nodes *A, B & C* must all be in range of each other, the future architecture is designed such that *A* and *B* can both be in range of *C*, while not within range of each other. Therefore, in this simple three-neuron example, essentially three sub-networks are formed, each centered about a particular neuron. They can be referred to as sub-nets, since the set of nodes participating in each are not identical. Such architecture would allow the geographical area covered by the network to increase without increasing the power requirements of nodes.

7.7 Conclusions

A new wireless sensor network architecture has been developed. A recursive neural architecture, RSOM-WSN offers several advantages over other state-of-the-art solutions. The architecture is self-organizing, tolerant of faults, and is synergistic with its hardware counterparts. Simple local interactions between nodes lead to complex global behavior, in which network configuration and parallel processing are achieved through a unified algorithm. In doing so, the proposed architecture is better tailored to specific inherent and desired characteristics of wireless sensor networks than other state-of-the-art architectures, which tend to treat wireless sensor networks as typical large-scale ad-hoc

networks. In developing RSOM-WSN, the work in this thesis has advanced the state-of-the-art in wireless sensor network architecture design.

References

[1] Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, "A survey on sensor networks", *IEEE Commun. Mag.* 40 (8) (2002) 102—114

[2] C. Chong and S. Kumar, "Sensor networks: evolution, opportunities, and challenges," Proc. IEEE, vol. 91, pp. 1247-1256, Aug. 2003.

[3] *10 Emerging Technologies That Will Change the World," MIT Technology Review.,* vol. 106, no. 1, pp. 33–49, Feb. 2003.

[4] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks," in Proc. ACM MobiCom '99, pp. 263–270, August 1999.

[5] Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, "A survey on sensor networks", *IEEE Commun. Mag.* Vol. 40, no. 8, pp. 102—114, 2002.

[6] Curino, C., et al., "TinyLIME: Bridging mobile and sensor networks through middleware", 3rd IEEE Int. Conf. On Pervasive Computing and Communications, pp. 61—72, 8-12 March 2005.

[7] Pottie, G.J., "Wireless Sensor Networks", Information Theory Workshop, Killarney, Ireland, pp. 139—140, 22-26 Jun 1998.

[8] Römer, Kay; Friedemann Mattern (December 2004). "The Design Space of Wireless Sensor Networks". IEEE Wireless Communications 11 (6): 54-61.

[9] J. M. Kahn, R. H. Katz, K. S. J. Pister, "Next Century Challenges: Mobile Networking for Smart Dust", ACM MOBICOM Conference, Seattle, WA, August 1999.

[10] Chi-Fu Huang, Yu-Chee Tseng, "The Coverage Problem in Wireless Sensor Networks", WSNA '03, September 19, 2003, San Diego, California, USA.

[11] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M.B. Srivastava, "Coverage problems in wireless ad-hoc sensor networks", IEEE INFOCOM, pages 1380–1387, 2001.

[12] X. Wang, G. Xing, Y. Zhang, C. Lu, R. Pless, and C. Gill, "Integrated Coverage and Connectivity Configuration in Wireless Sensor Networks," *Proceedings of Sensys*, 2003.

[13] Changsu Shu and Young-Bae Ko, "A Traffic Aware, Energy Efficient MAC Protocol for Wireless Sensor Networks", *IEEE International Symposium on Circuits and Systems, ISCAS 2005,* Vol. 3, pages 2975 – 2978, 23-26 May, 2005.

[14] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, John Anderson, "Wireless Sensor Networks for Habitat Monitoring", *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications WSNA '02*, September 2002.

[15] P. B. Chu, N. R. Lo, E. C. Berg, K. S. J. Pister, "Opti-cal Communication Using Micro Corner CubeReflectors", *Proc. of IEEE MEMS Workshop*, Nagoya, Japan, (January 1997), pp. 350-355.

[16] Chaiporn Jaikaeo, Chavalit Srisathapornphat, Chien-Chung Shen, "Querying and Tasking in Sensor Networks", *SPIE's 14th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Control*, 2000

[17] Chaiporn Jaikaeo, Chavalit Srisathapornphat, Chien-Chung Shen, "Sensor Information Networking Architecture and Applications", *IEEE Personal Communications,* Vol. 8, Issue 4, August 2001, pp. 52-59.

[18] Shah, R.C., Rabaey, J.M., "Energy Aware Routing for Low Energy Ad Hoc Sensor Networks", *IEEE Wireless Communications and Networking Conference*, Vol. 1, 17-21 March 2002, pp. 350-355.

[19] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks", *IEEE Proc. Hawaii Int'l Conf. Sys. Sci.*, Jan. 2000, pp. 1–10.

[20] Younis, O.; Fahmy, S.; "Distributed Clustering in Ad-Hoc Sensor Networks: A Hybrid, Energy-Efficient Approach", *INFOCOM 2004, Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 1, 7-11 March 2004, pp. 629-640.

[21] Ya Xu, John Heidemann, Deborah Estrin, "Geography-Informed Energy Conservation for Ad Hoc Routin", *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking MobiCom '01, ACM Press*, July 2001, pp. 70-84.

[22] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, Jorjeta Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols", *In Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, October 1998, pp. 85-97.

[23] David B. Johnson and David A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks", In *Mobile Computing*, edited by Tomasz Imielinksi and Hank Korth, chapter 5, pp. 153-181. Kluwer Academic Publishers, 1996.

[24] Perkins, C.E.; Royer, E.M.; "Ad-Hoc On-Demand Distance Vector Routing", *Second IEEE Workshop on Mobile Computing Systems and Applications, 1999. Proceedings WMSCA '99,* February 25-26, 1999, pp. 90-100.

[25] H. Zhang and A. Arora, "GS^3: Scalable Self-Configuration and Self-Healing in Wireless Networks", In *21st ACM Symposium on Principles of Distributed Computing (PODC) 2002.* pp. 58-67.

[26] H. Chan and A. Perrig, "ACE: An Emergent Algorithm for Highly Uniform Cluster Formation", *Proc. 1st Euro . Wksp. Sensor Networks*, Jan. 2004, pp. 154-171.

[27] Chou, J.; Petrovic, D.; Kannan Ramachandran; "A Distributed and Adaptive Signal Processing Approach To Reducing Energy Consumption in Sensor Networks" *INFOCOM 2003, Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE*, Vol. 2, 30 March – 3 April 2003, pp. 1054-1062.

[28] Oldewurtel, F.; Mahonen, P.; "Neural Wireless Sensor Networks", *International Conference on Systems and Networks Communication 2006, ISCNC '06*, Oct. 2006, pp. 28-37.

[29] Kulakov, A.; Davcev, D.; Trajkovski, G. "Implementing Artificial Neural-Networks in Wireless Sensor Networks", *2005 IEEE/Sarnoff Symposium on Advances in Wired and Wireless Communication*, April 18-19 2005, pp. 94-97.

[30] Caterterall, E., Van Laerhoven, K., and Strohbach M., "Self-Organization in Ad Hoc Sensor Networks: An Empirical Study", *In Proc. Of Artificial Life VIII, The 8th Int. Conf. on the Simulation and Synthesis of Living Systems,* Sydney, NSW, Australia, 2002.

[31] Freek Zindel, 2006. *A Neural Infrastructure for Wireless Sensor Networks.* Thesis (MSc). Delft University of Technology.

[32] Chapter 2 - The Role of Government in the Evolution of the Internet; Revolution in the U.S. Information Infrastructure; National Academy of Sciences; 1994.

[33] J.-S. R. Jang, C.-T. Sun, E. Mizutani, *Neuro-Fuzzy and Soft Computing*, Prentice Hall, 1997.

[34] Markus Varsta, Jukka Keikkonen, and Jose del Ruiz Millan. "Context Learning with the self organizing map". *In Proceedings of WSOM '97, Workshop on Self Organizing Maps*, Espoo, Finland, June 4-6, pages 197-202. Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland, 1997. ISDN 951-22-3589-7

[35] Timo Koskela et al., "Temporal Sequence Processing Using Recurrent SOM", *Second International Conference on Knowledge-Based Intelligent Electronic Systems,* Adelaide Australia, 21-23 April, 1998

[36] Varsta, et al., "Temporal Kohonen Map and the Recurrent Self-Organizing Map: Analytical and Experimental Comparison", *Neural Processing Letters, Springer Netherlands*, Vol. 13, No. 3, June 2001, pp. 237-251

Vita Auctoris

Matthew Ball was born December 26th, 1982 in Windsor, Ontario. He graduated from Honourable Vincent Massey Secondary School in the year 2001. He went on to receive his Bachelor of Applied Science degree in Electrical and Computer Engineering from the University of Windsor in 2005, where he is currently a candidate for the degree of Master of Applied Science, with the intent to graduate during the Fall semester of 2007.