Trade-offs in the Distribution of Neural Networks in a Wireless Sensor Network

Mike Holenderski *, Johan Lukkien *, Tham Chen Khong †
* Department of Mathematics and Computing Science,
Eindhoven University of Technology,
The Netherlands
† Department of Electrical and Computer Engineering,
National University of Singapore,
Singapore

***Abstract*— This article investigates the tradeoff between communication and memory usage in different methods of distributing neural networks in a Wireless Sensor Network. A structural approach is presented, categorized in two dimensions: horizontal and vertical decomposition. Horizontal decomposition turns out to be more attractive, due to high reuse of data present at the processor node. General properties of an alternative sematic approach are suggested theoretically allowing to dramatically increase efficiency.**

I. Introduction

The field of Wireless Sensor Networks is developing very rapidly with the main research done in routing [13], localization [18], [19] and data fusion. In this paper we deal with distributing supervised learning in a Wireless Sensor Network, or more generally distributing a neural network.

The idea of distributed learning is not novel. The research into this area has mainly focused on employing super computers [2], [3], [4] and optimizing the time needed to train the system. Due to the restriction of resources in wireless sensor networks, especially communication and memory [1], this approach is not applicable in this field. Some works take into account the communication aspect of wireless sensor network [9], [10]. They take the data-parallelization approach, however, they focus on minimizing the error when fusing the output of the distributed neural networks, stating strong assumptions on the actual training, but not dealing with it directly. They distinguish between parameterized [9] and non-parameterized models [10].

Other training algorithms have also been studied. [8] suggests distributing the Q-learning algorithm for reinforcement learning. Compared to supervised learning, it requires more memory (for storing the Q-tables and current policies). Another approach dealing with coordinating the reward function rather than the global state is presented in [11]. However, due to the nature of reinforcement learning it converges much slower than supervised learning.

User interaction, in terms of training set for supervised and rewards for reinforcement learning, is considered in [20].

In this paper we present three approaches to distributing neural networks: centralized, horizontal and vertical decomposition. Each of them presents a different distribution of workload among several processor nodes, trading off the amount of communication and the amount of memory required by each node.

We have chosen the backpropagation neural network for its simplicity and efficiency, which makes it a good candidate for wireless sensor networks.

In Section II we state the assumptions about the context, Section III states the problem, Section IV describes a structural decomposition followed by a generalization in Section V with experimental results in Section VI. Section VII presents an alternative semantic approach with conclusions in Section VIII.

II. Context

Our research is in the context of Wireless Sensor Networks. For the experiments we used a set of Mica2 motes [33]. These are small processor boards equipped with an ATmega128L processor, 4 KB of memory, sensors (Acceleration, Magnetic, Light, Temperature, Acoustic), RF radio and battery. They represent an instance of the more general idea of low-cost computing and communication devices.

There are several scarce resources. Since the motes have a very limited battery power supply, the greatest challenge lies in overcoming the energy constraint. Energy consumption is influenced by two main factors: processing and communication. The radio communication is considered the most expensive in terms of energy consumption. Unfortunately, solutions developed for other platforms [2], optimizing latency rather than volume of communication, are usually not applicable in this field. Hence communication is the factor we will try to optimize in this paper.

An other scarce resource is memory. Motes usually have few kilobytes of memory which makes it difficult to store large data structures. Memory is a peculiar resource as it is cheap when there is enough of it, but when too little, the cost rises drastically. We will therefore also optimize the maximum memory requirement of our solutions.

There is a tradeoff between memory and communication costs. The more memory is present at each mote, the more data can be computed locally and thus less communication is required.

III. Problem description

Given is a neural network comprised of L fully connected layers, with N neurons in each layer [5]. We distribute this neural network evenly among P processor nodes by assigning to each processor a partition of the weights set together with the corresponding neurons in such a way that each weight is mapped to exactly one processor (note that the neurons from the original neural network can be mapped to more than one processor).

The cost function we try to minimize is the maximum communication cost and memory usage per processor node, assuming the neural network is distributed evenly between nodes, i.e.

$$Cost = \max_{p \in P}(Cost_{comm}(p) + k\, Cost_{mem}(p))$$

where k represents the weight of memory cost relative to the communication cost.

The more nodes we have, the less memory is required per node. On the other hand, with more processor nodes, each node has to communicate with more other nodes. With this paper we try to gain insight into the tradeoff between the memory usage and communication cost per processor node in different ways of distributing a neural network.

IV. Structural approach

We take a constructive approach to partitioning a neural network, in our case building it up from a predefined atom, shown in Figure 1. The atom consists of one layer of weights,

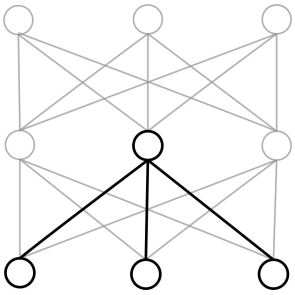

Fig. 1. Atom consisting of one upper neuron and $d = 3$ lower neurons, as part of a neural network

with 1 upper neuron and d lower neurons. The neuron in the upper layer is connected to all neurons in the lower layer and fuses the lower inputs by computing a weighted sum, and in case all inputs are present it applies the squashing function. We have chosen an atom to contain one output neuron due to this fusing nature, which is more efficient compared to the opposite situation with one lower and several output neurons.

Note that every neural network can be decomposed into a set of such atoms.

A. Construction

We can combine several atoms from the same layer in the original neural network to form a *horizontal* decomposition (see Figure 2). This can be seen as a generalization of an atom containing u upper neurons (instead of one) and d lower neurons. We can also stack several atoms on top of each other in l layers forming a *vertical* decomposition (see Figure 2). We can also combine both decompositions creating a network with l layers of u upper and d lower neurons. Thus we can categorize each decomposition of atoms into two dimensions: horizontal and vertical.[1]

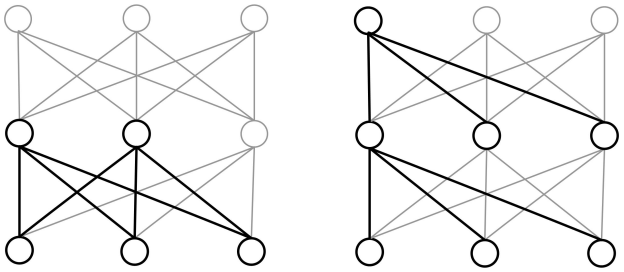

Fig. 2. Horizontal decomposition (left) and vertical decomposition (right)

Given is a neural network [2], comprised of L layers. A layer is a collection of neuron pairs connected by a weight, bounded by N lower and N upper neurons. Each upper neuron is connected to all lower neurons, and vice versa (as shown by the gray network in Figure 1).

There are N^2 weights in each layer, and thus LN^2 weights in the complete neural network. If we distribute the neural network evenly among P processor nodes, then for fixed L, N, and P, the maximum number of weights assigned to each node is

$$E_{max} = \left\lceil \frac{LN^2}{P} \right\rceil$$

We restrict ourselves to regular decompositions which assign to each processor node l layers with each having u upper neurons and d lower neurons. Thus the number of weights assigned to each node is ldu. Together with the above equation we have

$$ldu \leq E_{max}$$

Hence the number of upper neurons u of a layer is related to the number of lower neurons d and the number of layers per processor node l as follows:

$$u \leq \left\lceil \frac{E_{max}}{dl} \right\rceil$$

Now let us consider the communication process. We can split it into receiving (Rx) and transmitting (Tx) of inputs and outputs, respectively.

Listening to inputs coming from several other nodes requires keeping the radio listening for a certain time interval, which we assume to be proportional to the number of messages received.

[1]The two main methods of parallelizing neural network in the context of super computers, Pipelining and Neuron Parallelism (see [2]), can be seen as extreme cases of our horizontal and vertical decomposition, respectively.

Since we assume the original neural network is distributed evenly among the processor nodes, there are d lower neurons per layer per processor node. Each of these neurons needs N inputs, hence there are $\lceil \frac{N}{d} - 1 \rceil$ other processors nodes involved. Therefore each processor node must receive $ld \lceil \frac{N}{d} - 1 \rceil$ messages.

If $l > 1$ then some of the partial summations required by lower neurons of upper layers are computed on the processor node itself. This saves receiving $(l-1)\min(u,d)$ messages, where the $\min(u,d)$ takes care of the case when $d < u$.

For transmitting, since we are dealing with a radio transmitter, we can save on the communication by broadcasting the u outgoing outputs of all l layers to several nodes at once.

With this in mind we can formulate the communication costs as

$$\begin{aligned} Rx &= ld \lceil \tfrac{N}{d} - 1 \rceil - (l-1)\min(u,d) \\ Tx &= lu \\ Cost_{comm} &= Rx + Tx \end{aligned}$$

The memory usage $Cost_{mem}$, comprised of the number of weights and neurons (variables used to store the input and output values of the neurons), is given by

$$Cost_{mem} = E_{max} + l\max(d,u) + \min(d,u)$$

where the $\max(d,u)$ and $\min(d,u)$ take care of cases when $d < u$ [2].

Figure 3 shows the plot of the $Cost_{comm}$ function, with d on the x-axis and l on the y-axis, for a fixed N, P and L (note that u is directly related to l and d). We have omitted the ceiling function in the cost functions, in order to get a smooth line since we are interested in the tendency rather than exact values.

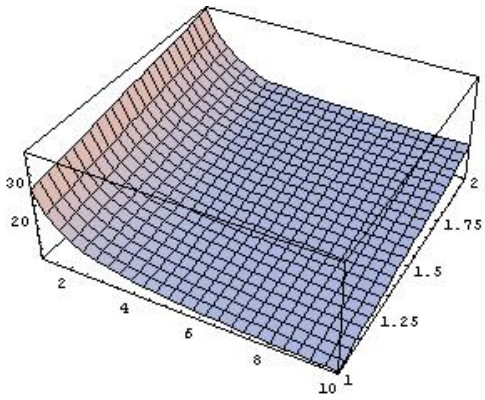

Fig. 3. Communication cost per processor note with $N = 10$, $P = 10$ and $L = 2$

The plot of the corresponding $Cost_{mem}$ function is shown in Figure 4.

[2]Check with Figure 2 by rotating the vertical decomposition upside down.

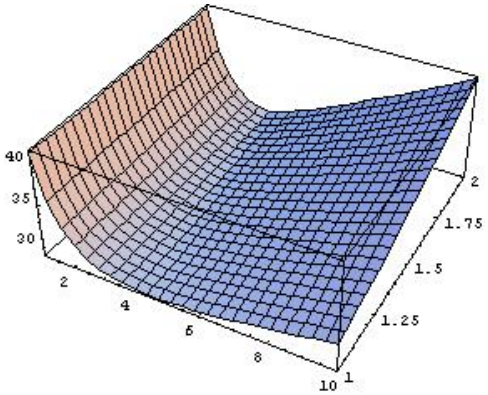

Fig. 4. Memory usage per processor note with $N = 10$, $P = 10$ and $L = 2$

B. Discussion

As we can see from Figure 3, the communication cost decreases with large d values (atoms containing many neurons in the lower layer compared to upper layer) and with small l values (flat horizontal decomposition). The advantage of horizontal decomposition can be attributed to the reuse of data present at the processor node. Once the input value for each lower neuron is received it is used by several upper neurons, as opposed to the vertical decomposition where each input is used only once.

Memory usage, as shown in Figure 4, is smallest when the upper and lower layers contain the same number of neurons. For $l = 1$ the minimum memory usage occurs at $d = 4$ where $u = 5$. This can be explained with weights sharing the neurons.

V. Generalization

Instead of dealing with atoms we can generalize the problem to distributing a neural network over several processor nodes by distributing the weights arbitrarily.

A. Cost function

If we consider feeding forward inputs trough a neural network, then each weight has an *in neuron* and an *out neuron*. Let $V_{in}(p)$ and $V_{out}(p)$ be the set of all *in neurons* and *out neurons* on processor node p, respectively. Again communication can be split into *Tx* and *Rx*.

On processor node p each neuron $n \in V_{in}(p)$ receives a value from each other node q, which computes a partial sum to n, i.e. where $n \in V_{out}(q)$. In case n belongs to the input layer of the original neural network then n receives a single value from the context (e.g. sensor). Thus, the Rx cost for node p is given by

$$Rx(p) = \sum_{q \in P \setminus \{p\}} |V_{in}(p) \cap V_{out}(q)| + |V_{in}(p) \cap V_{input}|$$

where V_{input} is the set of neurons of the input layer of the original neural network.

For Tx similar argument holds as for Rx, with the difference that if the value of an neuron $n \in V_{out}(p)$ is used by other

nodes then (due to broadcasting) it has to be sent only once. Thus

$$Tx(p) = |V_{out}(p) \cap \bigcup_{q \in P \setminus \{p\}} V_{in}(q)| + |V_{out}(p) \cap V_{output}|$$

where V_{output} is the set of neurons of the output layer of the original neural network.

The total communication cost is given by

$$Cost_{comm}(p) = Tx(p) + Rx(p)$$

The memory cost $Cost_{mem}$ again is the sum of all weights and neurons assigned to the processor node, thus

$$Cost_{mem}(p) = |V_{in}(p) \cup V_{out}(p)| + |W(p)|$$

where $W(p)$ is the set of weights assigned to node p.

Finding an optimal assignment of weights to processor nodes for given N, L and P presents an exponentially hard problem. We tried to approximate the solution employing a local search on the set of different distributions of weights.

B. Local Search

For the neighborhood function we chose a swap function exchanging two weights between two processor nodes. The cost of each neighbor is computed and the cheapest one is selected (greedy search). When the search gets stuck, several random swaps are performed to get the search out of a local minimum which is not a global minimum and the greedy search is continued. Initially the weights are distributed evenly between all processor nodes. Note that due to the swap neighborhood function the number of weights assigned to each node remains constant throughout the search.

C. Results

We have run the local search for different values of N, L and P, and a fixed value of $k = 0$ (i.e. focusing on the communication cost). Figure 5 shows a plot of the costs of the solutions against the degree of horizontal decomposition, for $N = 5$, $P = 4$ and $L = 2$. We measured the degree of horizontal decomposition by the product of the ratio between lower and upper neurons, and the average number of weights assigned to each layer.

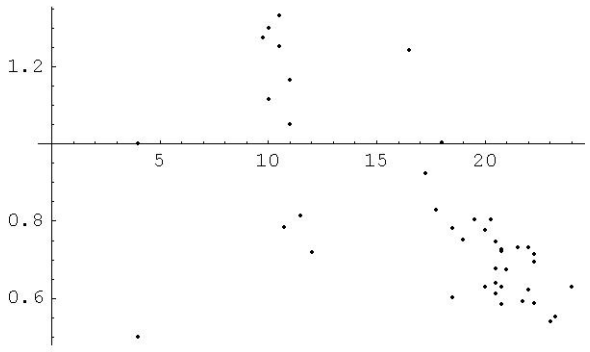

Fig. 5. Search results for $N = 5$, $P = 4$ and $L = 2$

The graph shows a trend with lower cost solutions having higher horizontal degree. The results confirm the conclusions from Section IV stating that flat horizontal decompositions are more cost efficient with respect to the cost function presented in Section III.

VI. Experimental Results

For testing we used the TinyOS operating system on a set of Mica2 motes. We have implemented a simple distributed event model where each mote can provide or use a service. For service discovery a simple flooding protocol was used, where a node broadcasts a request for a service upon which all nodes providing a matching service respond with a unicast message containing a description of the service. For the service data itself a broadcast was used as well.

In our test setup we had two *sensor* motes, two *brain* motes and one *actuator* mote. We arranged these in three layers: bottom layer containing the sensors, middle layer containing the brain (neural networks) and the top layer containing the actuator. We have implemented a fire alarm scenario as follows (see Figure 6):

- The two sensor motes provide a PHOTO service (they broadcast the value of the light sensor at regular intervals).
- The two brain motes use the PHOTO service and provide the FIRE service. They run a distributed neural network to learn the fire condition from the available PHOTO services and broadcasts a FIRE event if one occurs.
- The actuator mote uses the FIRE service. When it receives a FIRE event it sounds an alarm through a speaker.

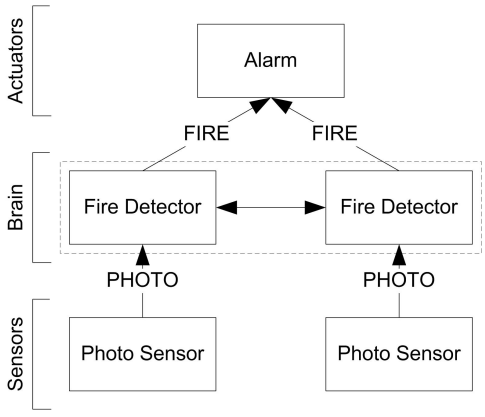

Fig. 6. Fire scenario setup

The brain motes collaborate and collectively determine the FIRE condition. We distinguished two cases:

1) centralized, where only one brain mote was used running the complete original neural network
2) distributed, where the original neural network was distributed between two brain motes according to the pipeline approach (flat horizontal decomposition).

The neural network contained 2 layers ($L = 2$), with 3 neurons per layer ($N = 3$). In both cases we employed supervised learning where the brain motes went through the stages of service discovery followed by training data acquisition, training and operation.

A. Discussion

Training a neural network required on average around 100 iterations, with 4 training patterns each. While this is feasible on a single mote running a centralized version of the neural network, it is not for the distributed version, due to the vast increase in communication (each training pattern has to be fed forward and backward between the motes). However, the described horizontal decomposition can be useful if one considers a neural network which is (partially) trained on a strong processor node and then distributed over weaker processor nodes and further trained, e.g. thorough reinforcement learning. The fact that the neural network is partially trained will reduce the communication requirement after deploying it on the processor nodes. With the advent of new technologies communication between sensor motes may become cheaper, making distribution of neural networks more attractive.

Both the horizontal and vertical decomposition present a structural approach to distributing networks. The set of weights of the original neural network is partitioned into smaller subsets which are then placed on different processor nodes. No information encoded in the weights is used. If one could find a more semantical approach, allowing to fuse some weights together, maybe one could find a more efficient solution.

VII. Sematic approach

One semantic solution suggests to remove weights having a value close to 0, since they affect the final result fairly little. We would like to take it a step further.

The approach of distributing neural networks presented in the previous section required coordination of partial results between processor nodes. Now we would like to come up with a way to deduce the partial results in higher levels from the knowledge of the weights in the original neural network and thus refrain from coordinating these in the lower layers.

The challenge is, given a trained neural network A, find a decomposition into smaller *independent* networks $a_1, a_2, \ldots, a_n$, which when merged together with some mapping M will compute an equivalent function to A, i.e. $M(a_1, a_2, \ldots, a_n) \equiv A$.

A. Example

Given a neural network A with input layer $x = (x_1, x_2, x_3, x_4)$, hidden layer $h = (h_1, h_2)$ and output layer $y = (y_1)$.[3] The layers are fully connected, with weights w_{lij} connecting node j from layer $l-1$ to node i in layer l . All neurons in the network compute the same squashing function $f : R \to R$.

[3]We abstract here from the $N \times N$ neural network for simplicity of discussion, without loss of generality [5].

In case of a linear function $f_{linear}(x) = ax+b$, the original network A computes function

$$A(x) = b+a(w_{231}(b+a(w_{111}x_1+w_{112}x_2+w_{113}x_3+w_{114}x_4))$$
$$+w_{232}(b+a(w_{121}x_1+w_{122}x_2+w_{123}x_3+w_{124}x_4)))$$

We can use distributivity of multiplication over addition to rearrange the terms in the above equation.

$$A(x) = b + abw_{231} + abw_{232}+$$
$$(a^2w_{111}w_{231} + a^2w_{121}w_{232})x_1+$$
$$(a^2w_{112}w_{231} + a^2w_{122}w_{232})x_2+$$
$$(a^2w_{113}w_{231} + a^2w_{123}w_{232})x_3+$$
$$(a^2w_{114}w_{231} + a^2w_{124}w_{232})x_4)$$

Since a, b and all weights w_{lij} are constant, we can write A as

$$A(x) = c_0 + c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4$$

where c_i are constants. Since each x_i in A occurs independent of any other x_i, A can be easily distributed among N nodes, by each node n_i sensing x_i and sending a single value c_ix_i to the central node, which then simply adds these together. Thus the decomposition is $a_i = c_ix_i$ with

$$M(a_1, a_2, \ldots, a_n) = c_0 + \sum_{i=1}^{n} a_i$$

□

Squashing functions are usually not linear. The typical function used in neural networks is the *sigmoid* function

$$f_{sigm}(x) = \frac{1}{1+e^x}$$

Note that f is bounded between 0 and 1. Each output of a two layer feed forward neural network computes a function of the form

$$f(\vec{h}) = f_{sigm}(w_1h_1 + \ldots + w_kh_H)$$
$$= f_{sigm}(w_1f_{sigm}(w_lx_1 + \ldots + w_mx_X) + \ldots$$
$$+w_kf_{sigm}(w_nx_1 + \ldots + w_ox_X))$$

where all weights w_i are constant. The challenge is to find a mapping M such that

- it will rewrite the above equation in such a way that each x_i occurs independent of any other x_i
- it minimizes the communication constraint

We do not restrict M to weighted summation and squashing function f, but allow any function, e.g. inverse of f, possibly involving more stages, as long as it can be computed on a mote in a reasonable time. Note that the decomposition has the knowledge of all the weights, i.e. they can be considered constants.

A picture of the mapping M is shown in Figure 7.

It should be possible to find M by working backwards from the original output function in y_1 and, using the knowledge of

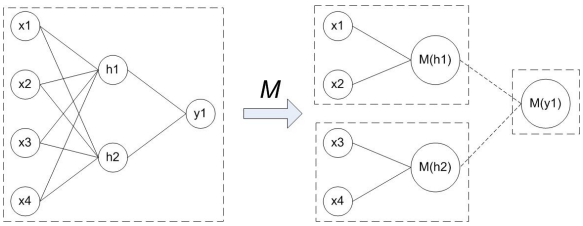

Fig. 7. Mapping M

all weights in the original neural network A, construct $M(h1)$, $M(h2)$ and $M(y1)$. Such an approach is similar to solving polynomial equations using Groebner Bases [32]. However, Groebner Bases operate on fields, which requires, among others, that the squashing function distributes over summation, but this is not the case due to the exponential in the sigmoid function.

VIII. Conclusion

As we have shown, distributing neural networks in wireless networks involves tradeoffs. As our results show, horizontal decomposition is more favorable than vertical, considering communication costs. Equal number of neurons in upper and lower layers results in optimal memory usage.

A semantical approach turns out to be a difficult challenge. More research has to be done into the algebra of neural networks, which if successful can prove to be much more efficient than the structural approach.

References

[1] J. Blumenthal, M. Handy, F. Golatowski, M. Haase, D. Timmermann, *Wireless Sensor Networks - New Challenges in Software Engineering*

[2] J. Torresen, S. Tomita, *A Review of Parallel Implementations of Back-propagation Neural Networks*, 1998.

[3] J. Torresen, S. Mori, H. Nakashima, S. Tomita, O. Landsverk, *Exploiting Multiple Degrees of BP Parallelism on the Highly Parallel Computer AP1000*

[4] F.M. Thiesing, U. Middelberg, O. Vornberger, *Parallel Back-Propagation for the Predistion of Time Series*, 1994

[5] H.M.M. ten Eikelder, *Neural Networks*, March 4, 2004

[6] Wil Michiels, Emile Aarts, Jan Korst, *The foundations of Local Search*

[7] I. Foster, C. Kesselman, S. Tuecke, *The Anatomy of the Grid*, 2001

[8] M. Lauer, M. Riedmiller, *An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems*

[9] J. B. Predd, S. R. Kulkarni, H. V. Poor, *Distributed leraning in Wireless Sensor Networks*

[10] J. B. Predd, S. R. Kulkarni, H. V. Poor, *Consistency in a Model for Distributed Learning with Specialists*

[11] P. Tadepalli, D. Ok, *Model-based Average Reward Reinforcement Learning*, 1998

[12] R. S. Sutton, A. G. Barto, *The Reinforcement Learning Problem*, MIT Press, 1998

[13] C. Intanagonwiwat, R. Govindan, D. Estrin, *Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks*, 2000

[14] R. J. Howlett, D.H. Lawrence, *A Multi-Computer Neural Network Applied to Machine-Vision*

[15] S.N. Simic, *A Learning-Theory Approach to Sensor Networks*, 2003

[16] Open GIS Consortium Inc., *Sensor Model Language*, 2004

[17] M. Leopold, *Tiny Bluetooth stack for TinyOS*, 2003

[18] F. Mondinelli, Z.M. Kovacs Vajna, *Self Localizing Sensor Network Architectures*, IEEE, 2004

[19] A. Savvides, H. Park, M.B. Srivastava, *The Bits and Flops of the N-hop Multilateration Primitive For Node Localization Problems*

[20] J.S. Sandhu, A.M. Agogino, A.K. Agogino, *Wireless Sensor Networks for Commercial Lighting Control: Decision Making with Multi-agent Systems*, 2004

[21] S. Maffioletti, S. Koudari M., B. Hirsbrunner, *Automatic Resource and Service Management for Ubiquitous Computing Environments*

[22] K. Seada, A. Helmy, *Rendezvous Regions: A Sccalable Architecture for Service Location and Data-Centric Storage in Large-Scale Wireless Networks*

[23] Golden G, Richard III, *Service Advertisement and Discovery: Enabling Universal Device Cooperation*, IEEE Internet Computing, 2000
Crossbow Technology, Inc., *Power Technology and Management*, 2004

[24] A.k. Dey, G.D. Abowd, D. Salber, *A Context Based Infrastructure for Smart Environments*

[25] Fahd Al-Bin-Ali, *A Design Philosophy for Inducing Reactivity in Intelligent Environments*

[26] N. Wallbank, *A Requirements Analysis of Infrastructures for Ubiquitous Computing Environments*, 2002

[27] G. Bieber, J. Carpenter, *Openwings: A Service-Oriented Component Architecture for Self-Forming, Self-Healing, Network-Centric Systems (Rev 2.0)*

[28] G. Bieber, J. Carpenter, *Introduction to Service-Oriented Programming (Rev 2.1)*

[29] Ang-Chih Kao, *Design and Implementation of Generalized Device Interconnect*

[30] M. Roman, R.H. Campbell, *Gaia: Enabling Active Spaces*

[31] E.A. Gryazin, *Service Discovery in Bluetooth*

[32] W. Adams, P. Loustaunau, *An Introduction to Grobner Bases*, 1994

[33] *MOTE-KIT 5x4x MICA2, MICA2DOT Professional Kit*, http://www.xbow.com