Distributed Event Localization and Tracking Algorithm (DELTA) Implementation

Masterarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Samuel Bissig
Juli 2009

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

Abstract

Different approaches to do event detection, tracking, localization and classification have been presented in the field of wireless sensor networks. As the nodes in wireless sensor networks have limited energy resources and low processing power, the used algorithms have to be efficient and energy aware. In this thesis we combine the advantages of different event detection, tracking and localization algorithms in a distributed event localization and tracking algorithm (DELTA). DELTA is extended with energy-efficient network management, event classification functionality and an energy based source localization. The energy-based network organization allows communication in a multi-hop environment with the base station. We present a distributed approach to build-up a backbone which connects all the nodes with the base station. Nodes which are not members of the backbone only wake up periodically to report sensor data and can so save energy. The algorithm is energy-based because nodes are not only elected as backbone node based on their position but also based on their energy level. DELTA presents a protocol to detect events and bundle information sensed by different nodes about one single logical event. In this thesis we present enhancements ,which allow DELTA to compute the position and intensity of an event based on the sensed amplitude. Further we present a self learning classification algorithm. Based on learned events, DELTA is able to learn event classes which can be used to classify unknown events.

Contents

Chapter 1

Introduction

1.1 Wireless Sensor Networks

Wireless Sensor Networks (WSN) have a wide application range. They are for example used for environmental or animal habitat monitoring, health-care, industrial process monitoring, applications in daily live and also military applications. In all these fields the main task of the WSN is to monitor physical or environmental conditions using the nodes of the network, which are equipped with arrays of sensors. Typically, these nodes communicate with a radio transceiver or with some other wireless communication device.

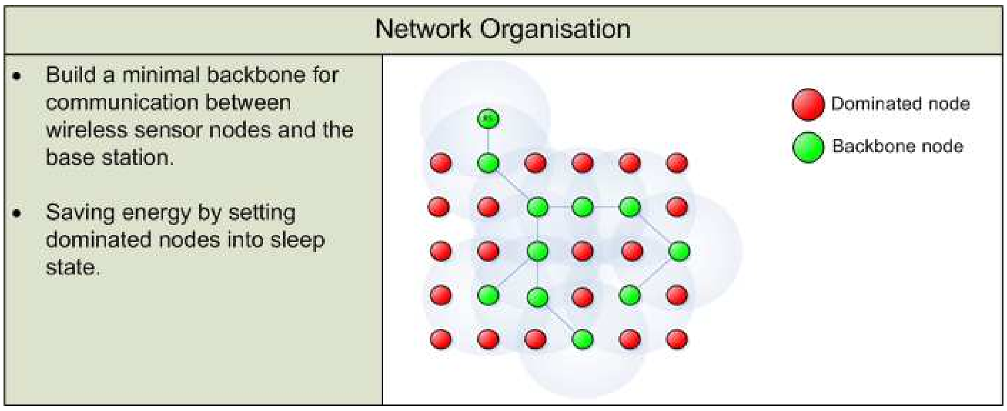
WSNs are self-organizing ad-hoc networks supporting multi-hop communication. Nodes can be distributed randomly. For example, they can be thrown out of a plane. In most cases, nodes which have been deployed in the environment, are difficult to be replaced. Consequently, the network must be remotely maintainable and robust in case of node failures. As WSN nodes commonly operate independently of a power supply, they are battery powered. Therefore, energy resources are a critical issue and nodes have to operate as efficiently as possible. This means that nodes are based on low power consuming hardware and also need efficient software, such as optimized network protocols.

1.2 Network Organization, Event Detection, Localization and Classification

In this work we present a distributed event detection, tracking, localization and classification framework (DELTA) [37], [38]. We present enhancements of the framework and evaluate them in real-world environments. The four main tasks addressed in this work are presented next.

Network Organization

In order to communicate in a multi-hop environment with the base station, to send some updates to the sensor nodes, or to report sensed data, communication paths are necessary. As communication is an energy consuming task, the network traffic needs to be minimized. In order to reduce energy consumption, the network nodes are switched off most of the time. As sleeping


Network Organisation

- Build a minimal backbone for communication between wireless sensor nodes and the base station.
- Saving energy by setting dominated nodes into sleep state.

Figure 1.1: Network organization in DELTA.

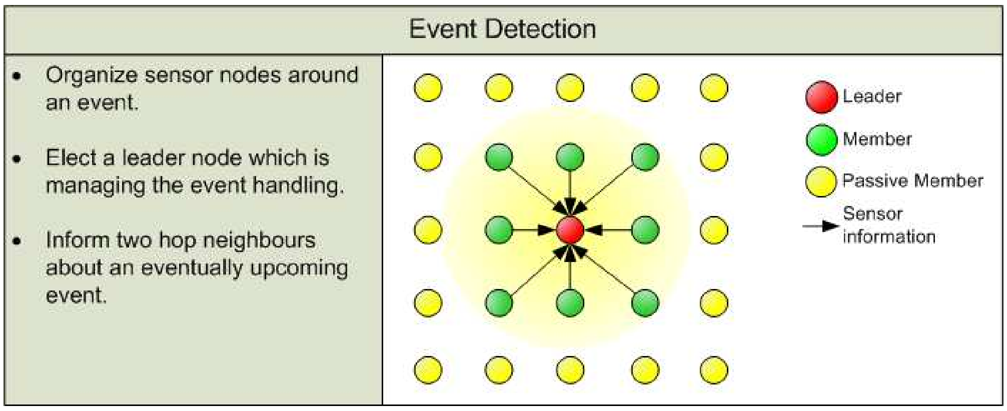
nodes cannot communicate and therefore are also not able to forward messages, some mechanisms to support stable communication paths have to be provided. Therefore, we present a network organization with two different kinds of nodes: Backbone nodes that provide a persistent communication path to the base station and non-backbone nodes which are sleeping most of the time and wake up periodically to send their sensed data to the base station over the backbone. In Figure 1.1 the green nodes are members of the backbone over which all data is routed. The red nodes are in a sleep state and wake up asynchronously to communicate with the base station. The roles are changed periodically to keep the energy on the same level over the whole network. Thus, a long network life time can be achieved. Finally, backbone nodes can move or fail. Therefore, a backbone repair mechanism is provided.

Event Detection

Events such as a vehicle are sensed by different nodes and different kinds of sensors on these nodes. Our WSN is able to bundle all information sensed by all nodes around an event source together to model one single logical event (see Figure 1.2). Upon sensing a measurement with a higher level than a predefined threshold, the nodes around an event elect a leader node, the red node in Figure 1.2, which will be responsible for the management of the sensed event. This manager node coordinates the distributed handling of the event. Neighbor nodes send their sensed information to the elected leader node for further handling. As events can be moving, the management node is responsible to perform the event tracking. Finally, the leadership is handed over to a node located closer to the moving event.

Event Localization

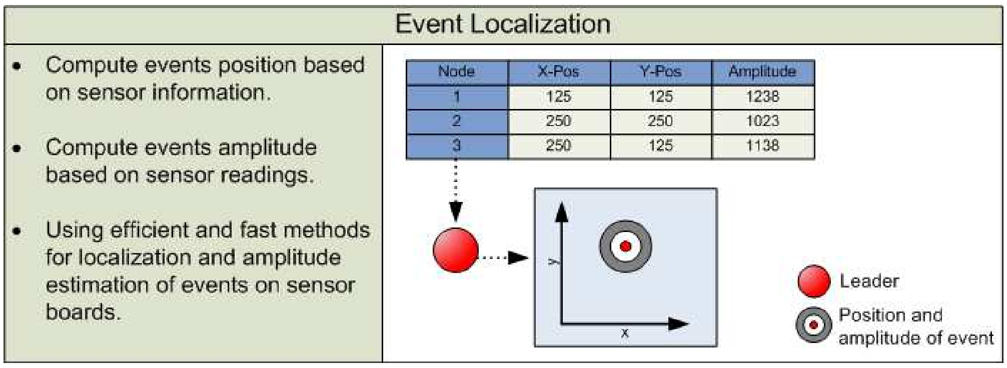
For some applications the event position needs to be known. We have implemented and evaluated different localization algorithms. Localization algorithms have to be fast and efficient, because computation power on WSN nodes is limited. In some cases not only the event position, but also other characteristics are interesting. Therefore, the presented algorithms also



- Organize sensor nodes around an event.
- Elect a leader node which is managing the event handling.
- Inform two hop neighbours about an eventually upcoming event.

Figure 1.2: Event detection and group organization in DELTA.

support the computation of the emitted signal strengths of the event. Figure 1.3 illustrates the event localization. The red node receives information about the event from its neighbour nodes. Based on this information it computes the events position and intensity. Therefore, a sensor model assuming isotropic signal attenuation is used.



- Compute events position based on sensor information.
- Compute events amplitude based on sensor readings.
- Using efficient and fast methods for localization and amplitude estimation of events on sensor boards.

Node	X-Pos	Y-Pos	Amplitude
1	125	125	1238
2	250	250	1023
3	250	125	1138

Figure 1.3: Event localization in DELTA.

Event Classification

A single event can be composed of different characteristics such as sound, vibration and light emissions. We present a simple distributed and self-learning classification algorithm to categorize these different characteristics into classes of events. We want, for example, to classify different vehicles in road traffic. Depending on the characteristics such as sound and vibration, we classify detected events into one of the classes *Bicycle*, *Motorbike*, *Car* or *Lorry* as shown in Figure 1.4. To update the classification rules on the network nodes we further provide a simple update mechanism.

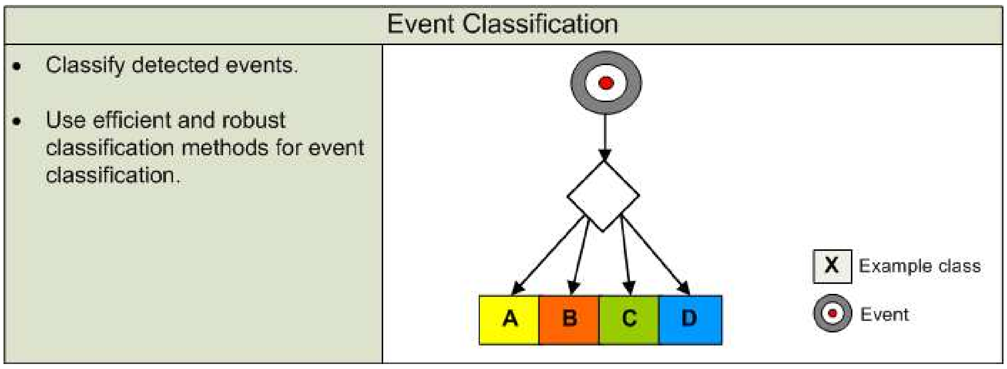

Event Classification
Classify detected events.
Use efficient and robust classification methods for event classification.
A
B
C
D
X Example class
Event

Figure 1.4: Event classification in DELTA.

1.3 Thesis Outline

We present related work on the topics addressed in this work in Chapter 2. We present approaches of Connected Dominating Sets to manage routing and network organization. Further, related work of event detection, localization and classification is presented. Chapter 3 introduces the enhancements to the existing DELTA framework done in this thesis. We have added localization and classification methods as well as a backbone mechanism to support communication between nodes and a base station. Chapter 4 describes implementation details of DELTA and its extensions. The real-world sensor node hardware is introduced. We describe how different parts of DELTA are implemented on these nodes. Chapter 5 provides the evaluation of the DELTA framework enhancements. We have implemented and evaluated the DELTA framework on a real-world sensor platform. Results of the evaluation from the different parts of the DELTA framework are discussed. Finally Chapter 6 concludes the presented work and discusses some topics for further investigations.

1.4 Contributions of this Thesis

This thesis is mainly build around the existing DELTA framework. In different areas DELTA has been extended. Mainly existing algorithms have been integrated into DELTA. They have been implemented and evaluated on real-world hardware. In the following we introduce the contributions in detail.

Network Organization

An existing algorithm for an energy efficient network organization based on connected dominating sets (CDS) was implemented on real-world hardware. Some modifications have been done due to fit the algorithm to the real-world environment. Different scenarios were constructed to evaluate the different parts of the algorithm on the real-world network. A framework for monitoring the distributed network communication was introduced.
To update the sensor nodes with new configuration settings we have extended the CDS algorithm

with a configuration distribution ability. Therefore no new data traffic has been introduced. The basestation informs the network nodes about new settings by piggy-backing this information to existing communication packets.

Event Localization

Different existing algorithms have been used to enable DELTA doing an appropriate localization and event strength estimation. After evaluating different algorithms the Simplex Downhill algorithm was chosen for an implementation on real-world hardware. DELTA with the extended localization abilities has been evaluated with real-world light emitting events.

Event Classification

To classify the estimated event locations and event amplitudes DELTA was extended by a classification framework. In a training phase the clustering algorithm K-Means is used to learn the event classes. In a second phase these information is used to classify the events during runtime. Two classifier algorithms are investigated. Finally, the Bayes classifier was chosen to be implemented on the real-world environment. The real-world implementation was evaluated with real-world light emitting events.

Chapter 2

Related Work

This chapter presents related work in the topics addressed later in this work. The first section deals with routing aspects. The theory of Connected Dominating Sets is introduced. In a second section different approaches for event detection, localization and classification are introduced. At the end we present the sensor hardware which has been used for the real-world implementation and evaluation.

2.1 Connected Dominating Sets

Energy savings and self-organizing topology control are important tasks of many WSNs due to limited energy resources and randomly deployed sensor nodes. Therefore, topology control and the construction of energy efficient virtual backbones have been largely investigated in ad-hoc and wireless sensor networks. The main focus of connected dominating set (CDS) approaches is to minimize the number of nodes in the backbone. This is referenced in literature as the ability of the algorithm to approximate the minimal connected dominating set (MCDS). This task is known to be NP-hard [3], though. There are different heuristics to approximate a MCDS backbone, which will be discussed in the following subsections, after a short introduction into the theory of CDS.

Let's have a graph $G = \{G, V\}$, where V is a set of vertices and E is a set of edges. A dominating set (DS) is a subset D of V such that every vertex not in D is connected to at least one member of D by an edge in E. In other words a DS of a graph G is a subset $V' \subset V$, where each node in $V - V'$ is adjacent to some node in V'. Figure 2.1 shows a DS, where the brown vertices are the nodes of D and the blue nodes are joined to at least one member of D. In a connected dominating set we have the constraint that the subset D needs to be connected. Figure 2.2 shows a CDS: The brown nodes are in D, the blue ones are joined to at least one member of D.

Different types of algorithms which make use of CDS to build a backbone in a wireless ad-hoc network, are presented in the next four subsections.

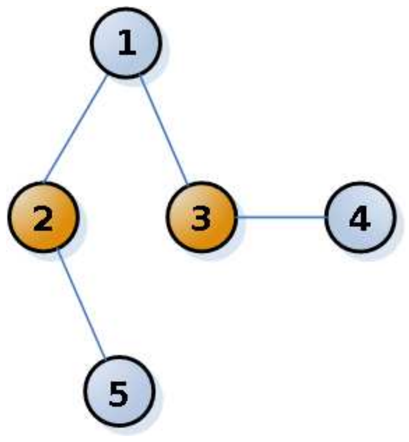


Figure 2.1: Dominating set.

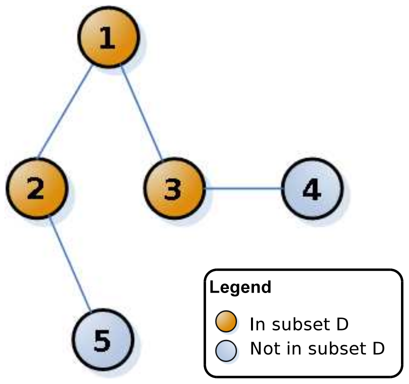


Figure 2.2: Connected dominating set.

2.1.1 CDS with Pruning Rules Approach

In [41] a simple distributed algorithm is proposed. It aims of building a CDS close to the MCDS. The algorithm supports changes in the underlying graph, which means that node failures or moving nodes are supported.

A marking process in an unweighted graph $G = (V, E)$ is proposed. *Unweighted* means that the edges have no labels or costs. A marker for a vertex v ∈ V can either be T (marked) or F (unmarked). Initially all vertices are unmarked. In the marking process and the subsequently applied pruning rules the following two terms are used: An *open neighbor set* $N(v)$ of a vertex v is the set containing all neighbors of v and is represented by $N(v) = \{u|\{v, u\} \in E\}$, where $\{v, u\}$ is the edge between vertex v and vertex u. A *closed neighbor set* $N[v]$ also contains v and is defined as $N[v] = N(v) \cup \{v\}$. In Figure 2.1 for example the set $\{1, 5\}$ is a open neighbour set of 2 and set $\{1, 2, 5\}$ is a closed neighbor set of 2. The marking process has three steps:

1. All vertices v get marked with F (unmarked).

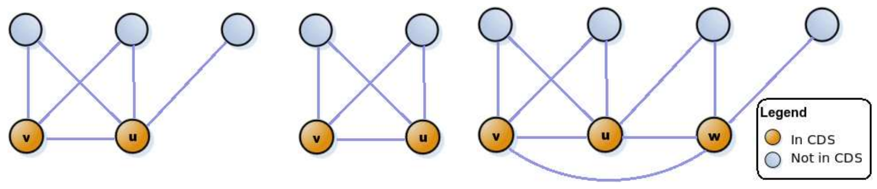
2. All vertices exchange their open neighbor set with their neighbors. After this step every node knows its two-hop neighborhood.

3. Every v which has two unconnected neighbors changes its marker to T (marked).

Using the example in Figure 2.2 we have the following open neighbor sets: $N(1) = \{2, 3\}$, $N(2) = \{1, 5\}$, $N(3) = \{1, 4\}, N(4) = \{3\}$ and $N(5) = \{2\}$. After step 2 of the marking process vertex 1 knows the neighbor sets of nodes 2 and 3, node 2 knows the sets of nodes 1 and 5, node 3 knows the sets of nodes 1 and 4, node 4 knows the set of node 3 and node 5 knows the set of node 2. In step 3 vertexes 1, 2 and 3 are marked with T, because they have two unconnected neighbors: node 3 has nodes 1 and 4, node 1 has nodes 3 and 2 and node 2 has nodes 1 and 5. Set V' is defined as the set of vertexes marked with T. The resulting set is not necessarily minimal. To reduce the size of the dominating set two pruning rules have been introduced.

Rule 1: We consider the reduced graph G' = G - V' and consider two vertices v and u $\in$ G'. For each node an unique numerical identifier $id()$ is introduced. It is used to avoid simultaneous removal of nodes from the CDS. If $N[v] \subseteq N[u]$ and $id(v) < id(u)$ then set the marker of v to F. This means that if the closed neighbor set of vertex v is covered by the set of vertex u, the vertex v can be removed from the CDS. In Figure 2.3 (a) v and u are CDS nodes and $N[v] \subset N[u]$, so the marker of v is removed from the CDS if $id(v) < id(u)$. In 2.3 (b) either v or u can be removed. To prevent removing both, the node with the smaller ID is chosen.

Rule 2: Vertexes v and w are marked neighbors of an also marked vertex u. If $N(u) \subseteq N(v) \cup N(w)$ and $id(u) = min\{id(w), id(u), id(w)\}$, the marker of u can be set to F. This means that if the closed neighbor set of a vertex is covered by the neighbor sets of two marked neighbors, this vertex can be removed from the CDS if the ID of this vertex is the minimum. In Figure 2.3 (c) $N(u) \subseteq N(v) \cup N(w)$. If $id(u) = min\{id(u), id(v), id(w)\}$, vertex u can be removed from the CDS based on Rule 2. If $id(v)$ was the minimum, vertex v could be removed based on Rule 1. The difference between Rule 1 and Rule 2 is that Rule 1 uses the open neighbor sets, while Rule 2 applies the closed neighbor sets.



(a) (b) (c)

Figure 2.3: Pruning examples. Rule 1 is applied in (a) and (b), rule 2 in (c).

Repair mechanisms for the following three cases have been presented: *mobile hosts switch on*, mobile *hosts switch off* and the *mobile hosts move*. The first two cases can be covered by the already known marking process and the two pruning rules. If nodes are moving the nodes can be updated locally. For this case a heartbeat signal is introduced which signals that nodes will start to move. Nodes in the backbone monitor this heartbeat signals after overhearing a start message. If they do not receive a heartbeat after a predefined interval, they determine a broken link to the moving host. If a node receives heartbeats without start signal, it determines a new link to a moving host.

The need of two-hop neighborhood information and poor performance in certain network topologies [35] makes the algorithm not appropriate for our propose.

In [40] an enhancement is presented which is based on the energy levels of the nodes instead of their link degrees. This supports longer lifetimes because only nodes with higher energy levels are elected into the backbone. In some cases non shortest paths are preferred to shortest paths containing nodes with poor battery levels. The rules defined in [41] are adjusted to get an energy-aware algorithm.

2.1.2 Maximal Independent Set Approach

Before explaining the algorithm we define an independent set (IS) and a maximal independent set (MIS). Given a graph G an IS is a subset of vertices V, so that none of these vertices in V is adjacent to another vertice. It is a MIS if adding any vertex to the set would break the independency property of the set. Consequently, any node not within the set must be adjacent to some node in the set in a IS as well as in a MIS. Figure 2.4 shows a graph with MIS. If one would remove one of the nodes in the MIS the set would be an IS.

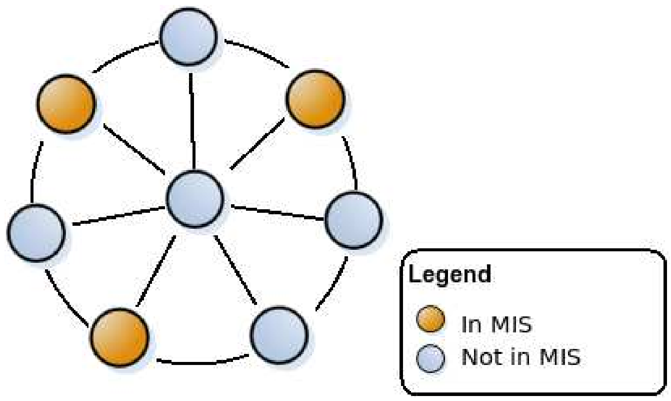


Figure 2.4: Example of a MIS.

[39] makes usage of a MIS in a first phase to build a CDS afterwards. The aim is to build a CDS with a low message complexity and good approximation of the MCDS. In the following we give a short conceptual overview over the CDS generating process.
In a first phase a MIS is build. First, a distributed leader-election algorithm such as presented in [2] is used to construct a rooted spanning tree. A communication protocol is proposed which is used to define the level for each node. The level of a node is defined as the number of hops to the root node of the tree. After this step all nodes know their own level and the levels of their neighbors. Based on this information a color-marking process is started which constructs the MIS.

In a second phase the root node initializes a process to connect the MIS to a CDS. In the presented approach only one-hop neighbourhood information is exchanged which leads to a message complexity of $O(n \log n)$. The algorithm has a good approximation factor in respect to an MCDS, but does not concern energy level of the nodes elected as CDS members. This would reduce the lifetime of the WSN. Furthermore, there is not mentioned any repair or update strategy. So CDS nodes which fail or move will not be replaced. Sleep cycles for non-CDS members are not supported. All these points make this algorithm inappropriate for our purpose.

2.1.3 Timer-Based Connected Dominating Set Construction

[42] presents a MAC-Layer Timer-based Connected Dominating Set Construction Protocol (MTCDS). The aim of the proposed algorithm is simplicity. The algorithm is optimized for IEEE 802.11 ad-hoc networks. No extra control messages are used, but all the messages are included in the beacons sent by the protocol. Changes in network topology such as failing or moving nodes, are supported. The algorithm is split into two phases: In the first phase an initiator is elected. In a second phase a CDS is constructed from the initiator node. The MTCDS protocol is based on a greedy strategy: The more uncovered neighbors a node has, the higher is its chance to be included into the CDS backbone. For the build process three states are proposed:

- $uncovered$: The node is not covered by a node in the CDS.
- $covered$: The node has a neighbor which is in the CDS.
- $inDS$: The node is in the CDS itself.

The initiator determines itself as member of the DS and starts the build process by broadcasting broadDS messages to its neighbors. Nodes in $uncovered$ state overhearing this message switch to $covered$ state and set a timer ΔT as described in equation 2.1.

$$\Delta T = T_{max} * \frac{1}{(\text{number of uncovered neighbors})^{\alpha}} \tag{2.1}$$

If this timer expires and the $covered$ node has not overheard a further broadDS, they set their state $inDS$, enters the DS and also starts broadcasting its $inDS$ state. Nodes in $covered$ state overhearing a broadDS message compete for the $inDS$ state only as long as they have any uncovered neighbors. The more uncovered neighbors a node has, the higher is the chance that the node switches into $inDS$ state.

The MTCDS can also adapt to node mobility. Nodes which fail, leave or are added or move are supported.

The proposed algorithm converges in only one algorithmic step. Thus, it is simple to implement, but it has a rather poor approximation factor. The energy distribution in the network is not concerned. The members of the CDS are elected based on their number of uncovered neighbors. Their energy level has no influence. As in all the other presented approaches, again no sleep cycles for non-CDS nodes are proposed.

2.1.4 Receiver-Based Backbone Construction

The goal of the approach presented in [35] is to support tracking and monitoring applications. The implementation of the protocol on real-world hardware and its evaluation is part of this thesis. Based on static or slowly moving nodes, the focus is on long-term lifetime. Therefore, the CDS adapts itself to local energy distributions in the network. Nodes not participating in the backbone shut down their radios and go to sleep for a predefined period, thus conserving energy. Primarily source-to-sink communication is supported. The algorithm builds a CDS by using periodically sent HELLO messages. Within these HELLO messages information about the one-hop neighborhood and on-demand also about the two-hop neighborhood is exchanged. The two-hop information is piggy-backed in the HELLO messages. The algorithm can be split in to three phases:

- Learning phase: The nodes learn their neighborhood.
- Setup phase: The backbone is constructed.
- Operation and maintenance phase: Data is sent from the sources to the base station. The CDS reacts on topology changes.

Learning phase

To learn its neighborhood each node sends HELLO messages in periodic cycles. A HELLO message contains also the actual state of the sending node. The state indicates if the node is already covered by the CDS or not. All nodes maintain a neighbor table containing neigborhood information. Neighbor nodes are all the nodes witch can be reached directly over a single-hop communication path. The table contains neighbor IDs, the state (dominator, dominated or none) of the neighbor and the timestamp of the last received message. The length of the learn phase is adapted according to the packet error rate. If it is high, the learning phase is longer to ensure that all nodes are able to learn their neighbors. It is set manually before distributing the sensor nodes.

Setup Phase

The base station is the only node which can start the network setup phase. Setting up the CDS is considered as a graph coloring problem. We first give a theoretical introduction into the setup phase and show the whole process according to an example afterwards. At the beginning all nodes are colored white. The network is setup as following:

1. The base station sets itself as dominator node and starts broadcasting DOMINATOR messages. These messages contain the neighbor table and the sender ID.
2. The nodes which receive the DOMINATOR messages check if they cover additional nodes by comparing their own neighbor table with the neighbor table of the DOMINATOR node. If they do cover any additional node, they go to sleep. If they cover any additional node they change to dominated state and start broadcasting DOMINATED messages. In these messages the neighbor table from the DOMINATOR message is forwarded, too. So the receiving nodes learn about the two-hop neighborhood.
3. The nodes two-hops away from the dominator node overhearing this DOMINATED message compare their own neighbor table with the one from the dominator node. Based on this information they prioritize their upstream neighbor as dominator and schedule a DOMINATOR_CHOICE message. This is the reason why the method is called *Receiver-Based* Backbone Construction. The priority of this message depends on the link-degree and the remaining energy of the dominator candidate. If there is only one known path

to the backbone, the DOMINATOR_CHOICE message is sent with the highest possible priority, without considering the remaining energy or link-degree of the dominator candidate. If there is more than one path to the dominator node, the priority depends only on the energy level of the sending node: The higher energy level of the sending node is, the higher is it prioritized.

4. Dominated nodes which receive a DOMINATOR_CHOICE message switch to dominator state and start also broadcasting DOINATOR messages.

An example is given in Figure 2.5. We assume that node D has received a DOMINATED message from node B and node E has received one from node C. As node D has no other path to the backbone than over node B, node B is elected as dominator. If node E will send a DOMINATOR_CHOICE before node D, node C might be elected as dominator too. This would lead to an additional dominator (C) which is not needed to guarantee connectivity. Therefore, node B has to be elected as dominator by node D immediately to prevent that node E elects node C as dominator.

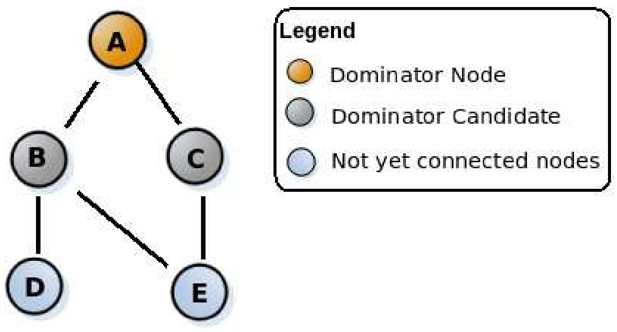


Figure 2.5: Example for short delay timer.

In Figure 2.6 the construction phase is depicted. Node A and B are neighbors of the base station. Node C is in the neighborhood of node B. The base station broadcasts a DOMINATOR message to its neighbors. Upon receiving the message, node A sets its state to dominated and enters sleep mode because it does not cover any other node. Node B, which has an uncovered neighbor, sets a timer to send a DOMINATED message. After the DOMINATED message has been sent to the neighbors of node B, node C sets a DOMINATOR_CHOICE timer. Node C has no other connection to the dominator, so the DOMINATOR_CHOICE timer is set with highest priority, which means with shortest possible delay. If node C would have other neighbors and if these neighbors would have an additional connection to the upstream dominator, node C would set a timer delay according to the energy levels of the dominator candidates.

Upon receiving the DOMINATOR_CHOICE message, node B joins the CDS, sets itself to the dominator state and broadcasts a DOMINATOR message. When node C receives this message it sets its state to dominated and enters the sleep phase because it covers no additional nodes. The algorithm terminates as soon as there is no uncovered node left. This means that all the nodes are connected in a CDS.

In the following paragraph some details about the timer settings are presented. For the release timers of the control messages DOMINATOR, DOMINATED and DOMINATOR_CHOICE the same contention time is used. For the HELLO messages a longer contention period is chosen. The shorter intervals for control message make the network construction phase faster. HELLO messages are sent with longer interval periods to reduce the number of packets which has to be sent by dominator nodes. The maximal duration of these intervals depends on the sensor network properties and needs to be configured by the operator. The timer for the DOMINATOR_CHOICE message is composed of the dynamically chosen delay determined by the priority of the backbone candidate. If a control message timer is set, the hello interval is interrupted until the control message has been sent. More details about the timer settings are presented in the implementation part of this work (see Section 4.2.1).

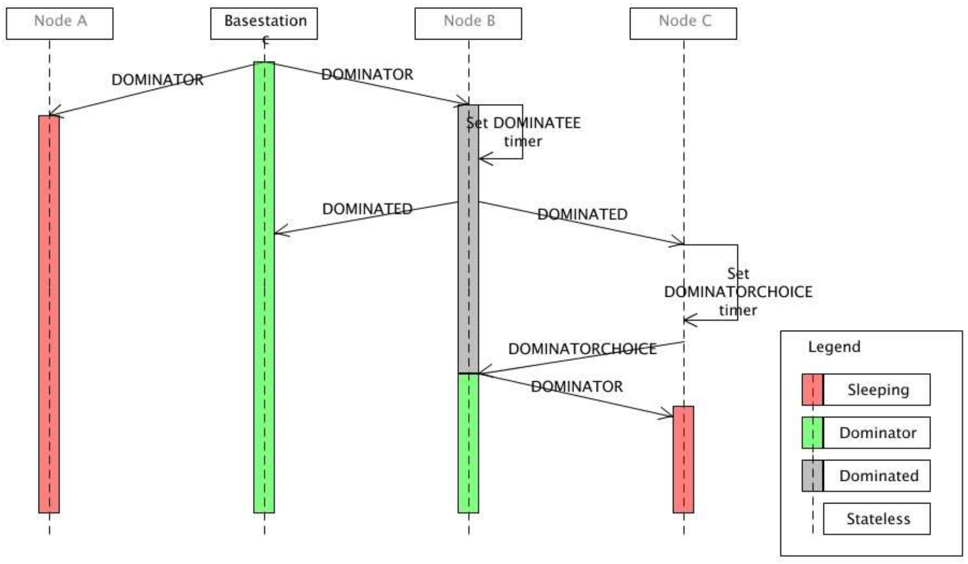


Figure 2.6: Sequence diagram CDS set up phase.

Operation phase

After the CDS has been established, the nodes keep their states for a predefined period called backbone time. After that the whole backbone is reestablished adapting itself to new network conditions and trying to keep the energy level uniformly distributed. During the backbone time the nodes follow a predefined listen/sleep cycle. In Figure 2.7 the cycles for a dominator and for a dominated node are shown schematically. The dominator nodes stay awake for the whole backbone time. The dominated nodes sleep mainly and listen to the medium periodically.

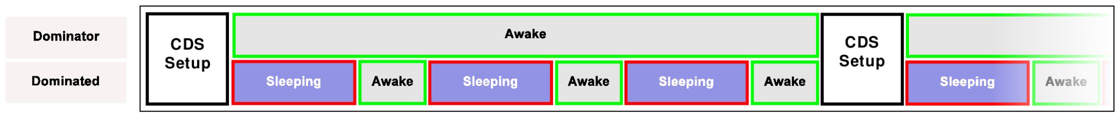


Figure 2.7: Different cycles during a CDS lifetime.

Path adaptation and repair mechanism

Our algorithm presented in [35] previews the following cases of network changes:

- A new node is added during run time.
- A dominated node is disconnected.
- A dominator node fails.

The first case is solved by just synchronizing new added nodes with the next backbone. If there is no backbone node within the nodes communication range, the node has to wait until the CDS network is reestablished. This is done periodically to adapt the network to the new energy levels of the nodes.

If a connection to a dominated node fails, this node is removed from the neighborhood table of a dominator node after a predefined timeout. If there was temporal link disruption and the node reappears it is again added to the neighborhood table.

If a dominator node fails, a link break is detected by one of its down-link nodes. The down-link node could either be the next dominator node in the path (green start point in Figure 2.8) or a neighboring dominated node (red start point in Figure 2.8). Link breaks are detected if the nodes do not overhear any hello message during a predefined time. Nodes which have detected a link break switch into a link break state (yellow in Figure 2.8) and start broadcasting link break messages to inform their neighborhood about the link break. Dominated nodes which overhear a link break message save the address of the link break node and forward the link break message. As soon as a backbone node with a valid route to the base station receives a link break message, it starts broadcasting link repair messages containing its valid path to the base station. Upon receiving such a link repair message, each node updates its path to the base station and forwards the message, with its ID added to the path, in its own link repair message. Dominated nodes rebroadcasting LR messages change their state to dominator nodes. To minimize the number of dominators after a link break, the path update distribution is done in contention. Nodes overhearing a link repair message from a node from which they have received a link break message before cancel their own path update procedure. They know that the link has been repaired by an alternative path.

Figure 2.9 shows an example of the repair mechanism. Dominated node A did not receive any hello message from its dominator. So it starts sending link break messages in its wake period.

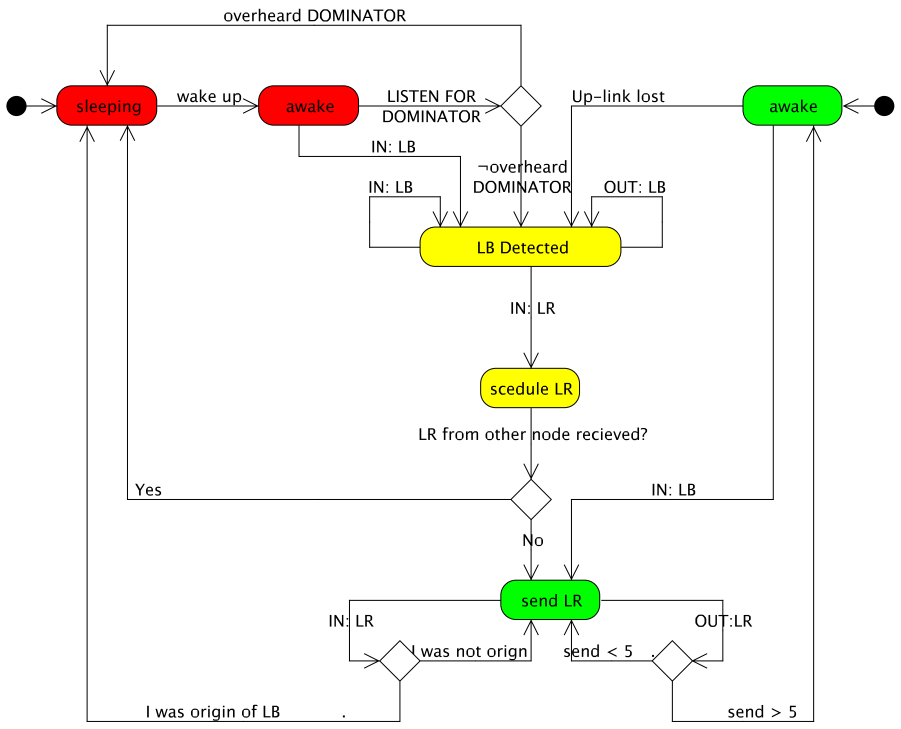


Figure 2.8: Sequence diagram of the repair algorithm.

Its dominated neighbors B and D overhear this message when they wake up. They forward the link break message. Dominator node C overhears this message and replies with a link repair message, which contains the path to the base station. Upon reception of the link repair message, node B changes its state to dominator. Then, it forwards the link repair message after adding its ID to the path to the base station. Finally, node A receives the link repair message and switches its state to dominator too. It forwards the link repair message. Node D does not change its state because it has overheard a link repair message from a node (A) from which it has overheard a link break message, before.

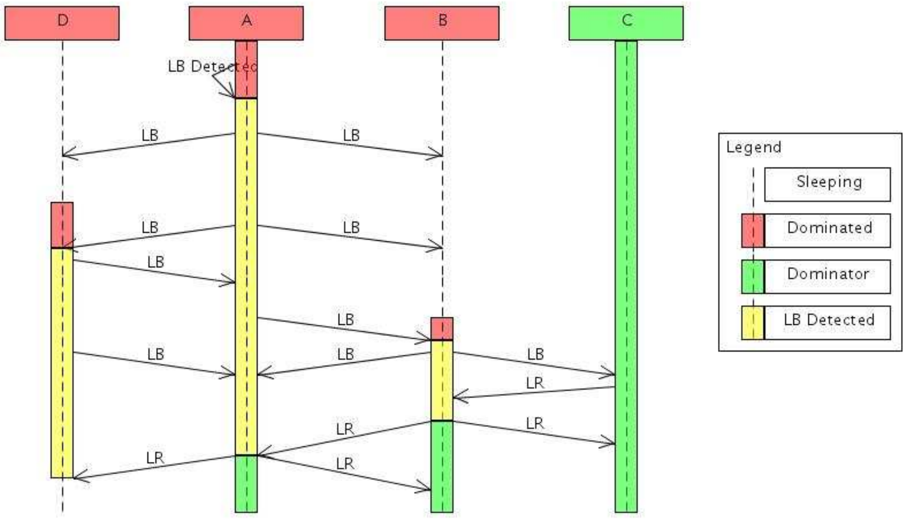


Figure 2.9: Example of the link break repair mechanism.

Two-hop neighborhood information is only distributed in the DOMINATOR and DOMINATED messages. This keeps communication and storage requirements low, while providing two-hop neighbors with relevant data considering the CDS setup process. This happens in a unidirectional way: The dominated nodes know the neighborhood table from the dominator node but not vice versa. During the operation phase, only one-hop neighborhood information is transmitted. The algorithm supports a long network lifetime, because the energy level is kept distributed over the network. By reestablishing the backbone periodically, not only the resources of a few single nodes are used, but from the whole network. A node with low battery level should not be elected as backbone node, as long as there is an alternative node. We have chosen this algorithm for a real-world implementation because it fits best our purpose. In simulations two different types of dominator prioritization have been evaluated: A selection depending on the link degree (CDS-LD) and a selection depending on the up-stream nodes energy level (CDS-E). We put our focus on the CDS-E approach in the real-world implementation.

2.2 Overview over Event Detection, Localization and Classification

There exist several approaches in event detection, localization and classification. After an introduction of some general aspects of event detection, localization and classification we present in a first section contributions from the networking and communication research field. Their focus is on optimizing network load. Often only coarse-grained tracking is supported. An accurate estimation of the events position is not possible. A classification of events is not possible as the approaches do not compute any characteristics of the events. In another section the focus is on approaches doing accurate localization and classification. These are mostly contributions from the collaborative signal processing (CSP) research field. Information from multiple sensor nodes is used to localize or classify an event. The price for the accurate localization and classification are higher network loads. Finally, we present the DELTA framework which tries to combine the advantages from both fields.

2.2.1 Event Localization Approaches

Localization based on sensor information can be done in different ways. The simplest way is to find the closest point of approach (CPA). The node with the highest measured amplitude is the node closest to the event. The location of this sensor node can be used as an estimate of the events location. CPA provides low accuracy, but is a fast and simple solution. CPA approaches are often used in event tracking.

Another solution is to make usage of the time difference of arrival of the signals at different sensors (TDOA). It is possible to compute an event position based on the time difference of arrival time of two signals at different sensors, the known position of the sensor nodes and the known speed of the signal propagation. By using this information, the position of the event can be computed using different existing algorithms such as PinPtr [14], which will be discussed later in this section. This method requires that the clocks of the nodes are synchronized, which causes additional network traffic.

A third possibility is to do energy-based source localization. This method is based on the fact that the energy level of the amplitudes of any kind of signal decreases with distance. In order to be able to use the sensor readings to accurately localize an event, adequate sensor models for the particular sensors are required. It is assumed that the sensor signals propagate isotropically (e.g. light, sound, vibration emitted by point sources). Such isotropic radiation models have been used in energy-based source localization ([32], [25] and [20]). The according sensor model is shown in Equation 2.2:

$$\rho_i = \frac{c}{||\mathbf{x} - \xi_i||^{\alpha}} + \omega \tag{2.2}$$

The received signal ρ_i at sensor node i at position ξ_i is inversely proportional to the emitted signal power c. ω is some additional white Gaussian noise and $||.||$ the Euclidean norm. α

depends on the kind of source which is sensed. It describes the attenuation degree of the emitted signal.

2.2.2 Distributed and Centralized Approaches

The presented contributions are either centralized or distributed. In the first case, sensor information is routed to a base station (BS) where the localization or classification is done. These approaches often use localization or classification algorithms with high complexity, which could not be performed on simple sensor nodes. Consequently, a base station with more resources and computing power is necessarily and a lot of network traffic is generated by routing all the sensed information to the base station. On the other hand distributed approaches often use less accurate and simpler algorithms for localization and classification. Network traffic can be reduced as only aggregated sensor information is routed through the network to the base station.

2.3 Event Tracking and Network Organization

The following approaches are mainly contributions from the networking and distributed communication research field. All contributions propose a kind of group organization and management of the nodes suited around an upcoming event, either in a dynamic or static way. Organizing nodes in local groups around the events reduces communication costs, because not the whole network is involved in the event handling.

2.3.1 Event Detection Using Static Sensor Clusters

In [43], after the deployment of the sensor nodes, static clusters are built. Each cluster has a cluster head which is expected to have more computational capabilities than the rest of the sensor nodes. The cluster heads are managing the clusters and do most of the computations. The target localization is organized by a two-step communication protocol. A sensor node sensing an event sends an event notification message to the cluster head. This is only a one bit message which indicates the appearance of an event. Detailed information like detection strength level is provided to the cluster head upon subsequent queries from the cluster head. Only the node which is assumed to be closest to the reported event is queried for more information. This method reduces network load as only selected events are reported to the base station. After the deployment of the sensor nodes the cluster head computes a table, which contains the probabilities of a sensor node sensing an event at a certain position. This table is used to choose the node which is queried for detailed information about the event.

The static cluster approach is not flexible. Events close to the cluster border could be reported by multiple cluster heads. This could lead to additional network load. The sensor network is not homogeneous as different node types are used which results in additional costs.

2.3.2 Coarse-Grained Localization

In [12] and [13] the localization of both node and event positions is proposed. In both approaches landmark nodes which know their absolute position are used. A distributed algorithm makes usage of geometric constraints induced by radio connectivity to estimate the nodes or events positions. The approaches use negative and positive connectivity information. Positive information reduces the location of the node or event to a region of finite size. If for example a node receives a message from a landmark node it knows that its own position must be somewhere in the estimated sending range of the landmark node. Negative information precludes a node or event from appearing in a certain region. In other words it is information about the position where a node or event cannot be located. [12] makes usage of rectangles to represent the possible node position, while [13] uses Bézier polygons, which supports also non-convex regions.

The accuracy of the algorithms is limited and the algorithms impose rather high delays. Classification of the events is not intended.

2.3.3 EnviroTrack

EnviroTrack ([1] [23]) is a middleware layer that exports a new address space to the sensor network. Not the sensors themselves but the physical events in the environment are the addressable entities. Objects are tracked by dynamically established groups of sensor nodes. EnviroTrack [1] implements a CPA based tracking and it offers a group management service. It organizes the nodes which are responsible for tracking an event and also determines if the event is moving. Figure 2.10 illustrates the group organization of EnviroTrack. Each of the event tracking groups has a single leader node. Nodes which are sensing an event start a leader election process. The leader is elected based on a randomly defined timeout. After the timeout has expired a message is broadcasted. The first node sending this message is the leader of the tracking group. Nodes which receive such a message and sense the same event become group members. EnviroSuite ([23]) has a slightly different leader election algorithm. The timeout for the leader message is not chosen randomly, but it is defined inversely proportional to the remaining battery level of a node. So nodes with higher energy level will have a higher chance to be elected. This prolongs system lifetime. Leader nodes continue to send heartbeat messages periodically. These messages inform the members that the leader is active. If a leader becomes inactive, a new leader election is started after a predefined timeout. Nodes which are not sensing the event, but overhear heartbeat messages, are called group followers. If the event is moving into the direction of a follower node, it has a high probability to start sensing the event soon too. So it joins the group instead of building a new group. Leaders getting out of sensing range send a leader handover message to initiate a new leader election.

EnviroTrack offers a simple and efficient algorithm. But there are also some limitations. It requires a communication radius larger than twice the sensing radius to prevent concurrent leaders. EnviroTrack offers only tracking. There is no localization or classification functionality.

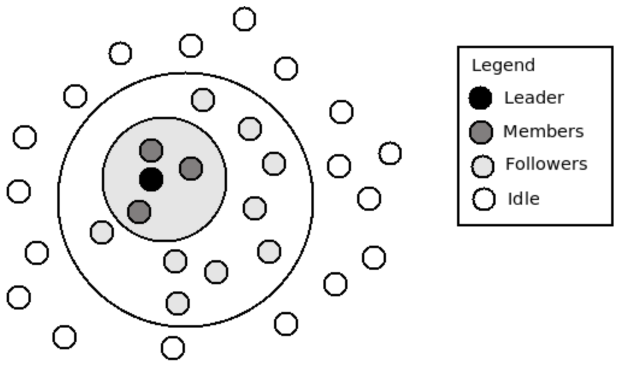


Figure 2.10: Group organization in EnviroTrack.

2.3.4 SensIt - Target Tracking

In the SensIt project [21] a framework for event detection, tracking and classification is presented. The WSN is split into dynamically established regions. Each region has at least one manager node. Event classification has been the main focus of the work and will be discussed in Section 2.4. The tracking of a target consists of five steps:

1. Nodes of a cell A (see Figure 2.11) detect a target. It is detected as soon as the sensor output exceeds a threshold. The detecting nodes report their measured energy levels to the cell manager nodes at *N* successive time instants.

2. The manager nodes compute at each time instant the location of the target using the received information. Therefore, an energy-based target localization algorithm has been proposed.

3. Manager nodes predict at the N time instants the location of the event at M ($M < N$) future time instants.

4. These predicted positions are used to create new cells that the event is likely to enter. The cells are built based on the velocity of the tracked target. In Figure 2.11 this is shown with the three dotted squares. The new cells are activated for subsequent detection of the event.

5. When the event enters one of the new cells, the old cell A is deactivated. The new cell takes over the control. Nodes in the old cell are set into standby state to conserve energy.

Information collected by manger nodes of cell A is passed to the managers of the next cell. This is important in case of tracking multiple targets. The presented handover mechanism involves rather high communication costs. A lot of nodes are involved when an event moves from one cell to another. The communication required within the cells is high too. Details about the event localization and classification aspects of SensIt can be found in Section 2.4.

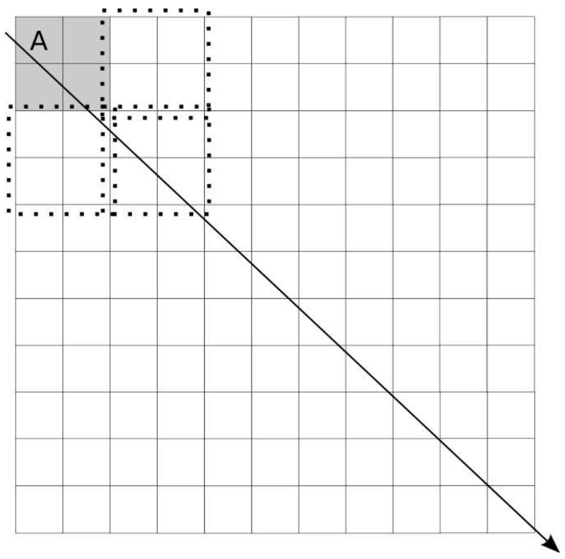


Figure 2.11: Event tracking using SensIt.

2.4 Event Localization and Classification

In the last section we have presented related work focusing on event detection and tracking and the organization of networks while events occur. In the following section we present related work which focuses on event source localization and classification. As mentioned in the introduction, most of the contributions in this section are from the CSP research field and focus on solving the localization and classification problems. Optimization of the communication load is often marginally considered. First we present some generic methods for localization and classification before presenting some more complex approaches.

2.4.1 Linear Least Square Method

To solve a system of equations of the form (2.2) with a standard linear least-square method, Equation (2.2) has to be transformed into a linear form. Assuming that the kind of signal is known, we also know the exponent α. If sound sources are sensed, then $\alpha = 2$ (see [18]). Equation (2.2) can be written as:

$$||x||^2 + ||\xi_i||^2 - 2x^T\xi_i - \frac{c}{\rho_i} \tag{2.3}$$

The noise parameter ω can be neglected, as it is regarded by over-determining the resulting system. From Equation (2.3) we remove the quadratic constraints on the unknown vector $\mathbf{x}$. Therefore we subtract the first equation ($i = 1$) from all the others ($i \neq 1$). N is the number of sensor nodes. The resulting system consists of $N - 1$ linear constraints of the following form:

$$2(\xi_1 - \xi_i)^T x + c\left(\frac{1}{p_1} - \frac{1}{p_i}\right) = ||\xi_1||^2 + ||\xi_i||^2 \tag{2.4}$$

Next we combine some terms in a vector $\hat{\mathbf{x}} = [x; c]$ and simplify Equation (2.4) by setting

$$a_i = \left[2(\xi_1 - \xi_i); \left(\frac{1}{p_1} - \frac{1}{p_i}\right)\right] \tag{2.5}$$

and

$$b_i = ||\xi_1||^2 + ||\xi_i||^2 \tag{2.6}$$

Substituting Equation (2.5) and Equation (2.6) in Equation (2.4) we get a simplified equation $a_i^T \hat{\mathbf{x}} = b_i$. Considering all N-1 linear constraints, we can write the system in matrix form as $\mathbf{A}_{N-1}\hat{\mathbf{x}} = \mathbf{b}_{N-1}$. Thus, we have got a system which can be solved in closed-form with a standard Linear Least Square method $\hat{\mathbf{x}} = (A^T A)^{-1} A^T b$. The number of sensing nodes has to be at least one element larger than the problem dimension. However, to obtain useful results an over-determined system should be used.

2.4.2 Nonlinear Methods

In order to use nonlinear optimization methods, we have to reformulate Equation (2.2) as a nonlinear least-square objective function:

$$f(\mathbf{x}, c) = \sum_{i=1}^{k} \left(\rho_i - \frac{c}{||\mathbf{x} - \xi_i||^\alpha}\right)^2 \tag{2.7}$$

Using Simplex Downhill or Conjugate Gradient Descend this equation can be minimized. This means that Simplex Downhill or Conjugate Gradient Descend are used to search an input value for which Equation (2.7) results in a minimal output. ρ_i is the measured sensor reading and $\frac{c}{||\mathbf{x} - \xi_i||^\alpha}$ is the model. This means the minimum error between the sensed value and the according model is determined.

Simplex Downhill

Simplex Downhill [26] works without derivations, it only uses function evaluations. The solution to the problem is found iteratively by searching a minimum in a multidimensional function space. As the name of the algorithm says it is based on simplexes. A simplex is a form with the simplest volume in a N-dimensional space. If $N = 1$ it is a line, for $N = 2$ it is a triangle and so on. For each point of the simplex a function value is computed. The function value can be seen as cost or error. From the $N + 1$ resulting values the lowest and highest values are identified. In each iteration the highest value gets replaced. This is done by geometrical operations. The highest point is mapped to a point with lower costs using either reflection, reflection and expansion, contractions or multiple contraction (see Figure 2.12).

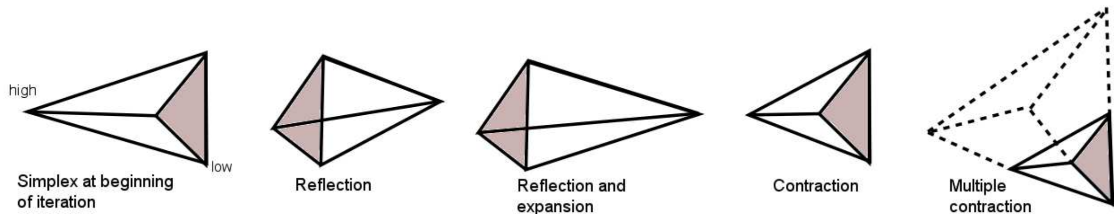


Figure 2.12: Possible geometrical operations in Simplex Downhill.

Using these transformations the Simplex Downhill is able to approximate any local or global minimum in the multidimensional function space. The termination criterion of the algorithm is defined by the size of the simplex. If it falls below a defined threshold Simplex Downhill terminates. As with all nonlinear optimization algorithms, it is also possible that Simplex Downhill finds a local minimum. The choice of the starting point has great influence on that. There exist some additional search procedures for finding a global minimum such as Monte Carlo methods. These methods are very cost-intensive and are not applicable on hardware with low computing power such as sensor nodes.

Conjugate Gradient Descent

Conjugate Gradient Descent [27] is an iterative method which, in contrast to Simplex Downhill, is using derivations. The problem to be solved is described as a N-Dimensional point $\mathbf{P}$. For these point values $f(\mathbf{P})$ and also the gradients ($\nabla f(\mathbf{P})$) have to be computed. Conjugate Gradient Descend is similar to the Steepest Descent method. This method starts from a point P_0. As many times as needed the algorithm move from point P_i to the point P_{i+1} in the direction of the local downhill gradient defined by $-\nabla f(\mathbf{P})$. In many cases the Steepest Descent method is not efficient and needs many steps to terminate. Conjugate Gradient Descend improves this method by computing the direction of the descent in a more sophisticated way. Instead of using the local gradient it uses the conjugate directions for going downhill. Figure 2.13 shows two example paths of a Steepest Descent and Conjugate Gradient Descent. After each iteration both algorithms define the direction and the distance they want to go. As we can see, Conjugate Gradient Descend converges in fewer steps than the Steepest Descent method. Like Simplex Downhill also Conjugate Gradient Descend can terminate in a local minimum.

2.4.3 Learning Event Classes with K-Means

K-Means [11] is a popular clustering algorithm. It is simple to program and apply. It takes the computed event amplitudes and the expected number of clusters as input and computes the cluster centers. It does not need any user interaction for computing the clusters. K-Means is searching exactly one splitting $C = \{C_1, ..., C_K\}$ of the given data set $S = \{x_1, ..., x_M\}$ where C_i are the clusters and x_i are the given sensed samples. The expected number of clusters K is

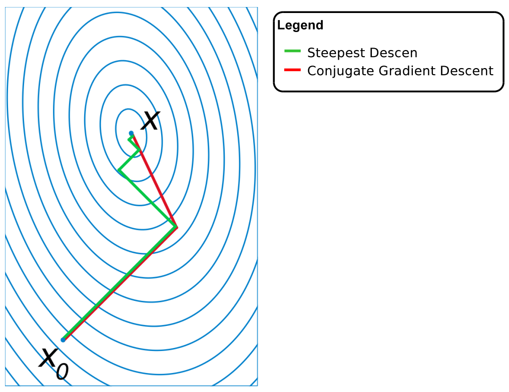


Figure 2.13: Comparison of the Steepest Descent and the Conjugate Gradient Descent (Figure: Oleg Alexandrov).

given by the user and does not change during the computation of the clusters. The following pseudo code 1 shows the basic functionality of K-Means:

```
Input: $S = \{x_1, ..., x_M\}$; $K$ = The number of clusters to find
Output: Clusters $C = \{C_1, ..., C_K\}$
Choose $K$ initial cluster centers $m_1, ..., m_K$ repeat
    Compute for each $x_i$ the closest cluster center;
    Compute for each cluster $C_i$ the costs;
    Recompute each cluster center $m_j$;
until Exit condition fulfield ;
```

Pseudocode 1: Psuedocode of K-Means.

There are different exit conditions possible:

- The cluster centers do not or only marginally change from one iteration to the other.
- The maximum number of iterations is reached.
- The number of patterns x_i which change the cluster from one iteration to the other is small.
- The costs are lower than a threshold or do not change from one iteration to an other.

The resulting clusters $C = \{C_1, ..., C_K\}$ depends on the initially chosen cluster centers $m_1, ..., m_k$. The solution is not necessarily an optimal one. Different strategies to choose the initial cluster centers are possible:

- Randomly choose K elements from S.

- Randomly generate K elements which are in the same n-dimensional cube as the elements from S.
- Use both mentioned methods above with the following additional constraint: A minimal distance has to be between the chosen cluster centers K.

There exist also optional post processing methods to improve the K-Means clustering. A method is to survey the compactness of the resulting clusters or to split clusters with a large variance. Methods for K-Means that guess the appropriate number of clusters for a specific K-Means clustering have been presented in [17]. We do not need such techniques in our implementation because the number of clusters is known in advance.

A disadvantage of the K-Means algorithm is the already mentioned problem with the initial cluster centers. The choice of the initial cluster centers influences the resulting clusters. By analyzing the training sets at the base station appropriate cluster centers can be determined in our application, though.

2.4.4 Minimum Distance Classifier

The minimum distance classifier is an intuitive and simple approach. The cluster centers C_i computed by the K-Means clustering algorithm are used. An event is represented by a feature vector $\mathbf{x}$. The components of $\mathbf{x}$ could be estimated with a technique such as presented in the previous section. For any sensed event the Euclidian distance D_i between the cluster center $\mathbf{z_i}$ of cluster C_i and the event $\mathbf{x}$ is computed. We get the following classification rule:

$$\mathbf{x} \in C_i \Leftrightarrow D_i(x) < D_j(x) \text{ for each } j = 1, ..., m; j \neq i \tag{2.8}$$

If there is no unique minimum a random sample is chosen as closest neighbor. The Euclidian distance can be simplified as we are interested in the relative value rather than the distance. So the square Euclidian distance is used:

$$D_i^2(\mathbf{x}) = ||\mathbf{x} - \mathbf{z}_i||^2 = \mathbf{x}'\mathbf{x} - 2\mathbf{x}'\mathbf{z}_i + \mathbf{z}_i'\mathbf{z}_i \tag{2.9}$$

The term $\mathbf{x}'\mathbf{x}$ is independent of class C_i, so it can be neglected and Equation 2.9 is simplified to:

$$d_i^2(\mathbf{x}) = -2\mathbf{x}'\mathbf{z}_i + \mathbf{z}_i'\mathbf{z_i} \tag{2.10}$$

Consequently, Equation 2.8 is equivalent to Equation 2.11:

$$\mathbf{x} \in C_i \Leftrightarrow d_i^2(x) < d_j^2(x) \text{ for each } j = 1, ..., m; j \neq i \tag{2.11}$$

2.4.5 Bayes Classifier

The Bayes Classifier [11] uses, in contrary to the minimal distance classifier, statistical and probability methods for classification. To train a Bayes classifier the membership of an element to a class has to be known. The training data is generated by the K-Means clustering algorithm. Important values and formulas which are used in the following are:

- $p(C_i)$: This is the probability of appearance of class C_i.
- $p(C_i|\mathbf{x})$: Is the likelihood of C_i given a sample $\mathbf{x}$.
- $p(\mathbf{x}|C_i)$: Is the likelihood of $\mathbf{x}$ given a class C_i.
- The Bayes theorem to get $p(C_i|\mathbf{x})$:

$$p(C_i|\mathbf{x}) = \frac{p(\mathbf{x}|C_i) * p(C_i)}{p(x)} \tag{2.12}$$

The Bayes classifier assumes that each assignment of an element to a certain class causes costs. One aim of the Bayes classifier is to minimize these costs. L_{ij}, with $L_{ij} >= 0$ are the costs to classify an element in a wrong class C_j while class C_i was the correct class. The mean costs $r_j(x)$ if a given $\mathbf{x}$ is classified into C_j having m classes are:

$$r_j(\mathbf{x}) = \sum_{i=1}^{m} L_{ij} p(C_i|\mathbf{x}) \tag{2.13}$$

To minimize the mean costs $r_j(\mathbf{x})$ the following classification rule can be applied:

$$\mathbf{x} \in C_i \Leftrightarrow r_i(\mathbf{x}) < r_j(\mathbf{x}) \text{ for } j = 1, ..., m; j \neq i \tag{2.14}$$

The mean costs r_i are computed for an unknown feature vector $\mathbf{x}$ in respect to each class C_i. Because $p(C_i|\mathbf{x})$ in Equation (2.13) is not known in most cases , it can be substituted with $p(C_i)$ and $p(\mathbf{x}|C_i)$ using Equation (2.12):

$$r_j(\mathbf{x}) = \frac{1}{p(\mathbf{x})} \sum_{i=1}^{m} L_{ij} p(\mathbf{x}|C_i) p(C_i) \tag{2.15}$$

The term $\frac{1}{p(\mathbf{x})}$ is independent of the class C_i so it can be discarded. We define the costs L_{ij}. If an element is assigned to the correct class, the costs L_{ii} are 0. In the other case the costs are $L_{ij} = 1$ ($i \neq j$). Inserting this values into Equation (2.15) we get:

$$r_j(\mathbf{x}) = \sum_{i=1}^{m} p(\mathbf{x}|C_i) p(C_i) - p(\mathbf{x}|C_j) - p(C_j) \tag{2.16}$$

Equation (2.16) can be simplified. We know that

$$\sum_{i=1}^{m} p(\mathbf{x}|C_i) p(C_i) = p(\mathbf{x}) \tag{2.17}$$

So, the classification rule can be written as:

$$\mathbf{x} \in C_i \Leftrightarrow p(\mathbf{x}|C_i) p(C_i) > p(\mathbf{x}|C_j) p(C_j) \text{ for } i, j = 1, ..., m; j \neq i \tag{2.18}$$

We have now a classification rule which assigns a pattern $\mathbf{x}$ to a class C_i with minimal costs. The only elements which need to be known are $p(C_i)$ and $p(\mathbf{x}|C_i)$ of each class C_i. If the $p(\mathbf{x}|C_i)$ are normal distributed, we get in the case on n- dimensions ($2 \leq n$):

$$p(\mathbf{x}|C_i) = \frac{1}{(2\pi)^{\frac{n}{2}}} \exp\left[-\frac{1}{2}(\mathbf{x} - \mathbf{m}_i)'\mathbf{K}_i^{-1}(\mathbf{x} - \mathbf{m}_i)\right] \tag{2.19}$$

$\mathbf{m}_i$ and $\mathbf{K}_i$ are defined as following:

- $\mathbf{m}_i = E_i\{x\}$ is the mean of class C_i.
- $\mathbf{K}_i = E_i\{(\mathbf{x}-\mathbf{m}_i)(\mathbf{x}-\mathbf{m}_i)'\}$ is the covariance matrix of class C_i. $|K_i|$ is the determinant.

Instead of using $p(\mathbf{x}|C_i)p(C_i)$ we use a monotone function to reduce $p(\mathbf{x}|C_i)$. As we are not interested in the absolute values but only in the relations between a $p(\mathbf{x}|C_i)p(C_i)$ and a $p(\mathbf{x}|C_j)p(C_j)$ we define:

$$D_i(\mathbf{x}) = \log\left[p(\mathbf{x}|C_i)p(C_i)\right] \tag{2.20}$$

Classification rule (2.18) can now be written as

$$\mathbf{x} \in C_i \Leftrightarrow D_i(\mathbf{x}) > D_j(\mathbf{x}) \text{ for } j = 1, ..., m; j \neq i \tag{2.21}$$

Finally we substitute $p(\mathbf{x}|C_i)$ in Equation (2.21) with Equation (2.19) and get the final decision function:

$$D_i(\mathbf{x}) = \log(p(C_i)) - \frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\mathbf{K}_i| - \frac{1}{2}(\mathbf{x} - \mathbf{m}_i)'\mathbf{K}_i^{-1}(\mathbf{x} - \mathbf{m}_i) \tag{2.22}$$

The summand $\frac{n}{2}\log(2\pi)$ can be neglected as it is independent from class C_i.
If only one dimensional input data is available, Equation 2.22 can be simplified.

2.4.6 SensIt - Target Localization and Classification

In the SensIt project work in event localization and classification has been done. Parts of the project are based on an energy decay model. The developed algorithms make usage of one single signal. In the following paragraphs we present different approaches which have been used in the SensIt project for classification and localization.

In [21] seismic signals have been localized and classified. As described in Subsection 2.3 the network is divided into groups with management nodes. Localization and classification are performed at these management nodes. The seismic signal is modeled as shown in Equation 2.2. To compute amplitude and position of the event, a system of Equations of the form 2.2 is solved with a linear least square (LLS) method. To classify the resulting values three classification methods are explored: A *k*-nearest neighbor (*k*NN) classifier, maximum likelihood (ML) classifier and a support vector machine (SVM) classifier. The presented algorithms operate on time series of measurements associated with detected events. This fact leads to classification delays as data has to be collected over a time period. It also sets high requirements on computational power and memory storage. In [29] and [30] refinements of these statistical methods have been presented.

In [19] methods for single event localization have been investigated. Four nonlinear optimization methods have been presented: Exhaustive Search (ES), Nelder and Mead Simplex Downhill, Conjugate Gradient descent method and Multi Resolution (MR) search. The performance of ES is worst, Simplex Downhill, MR and GD have approximately the same search complexity. All these methods face the problem of finding local optimums. To minimize this risk, the feasible solution space is overlaid by a grid, whereof the respective optimization procedure is performed at each point. The conclusion in [19] is to apply ES on a coarse grid in a first step and apply one of the better performing algorithms (Simplex Downhill, Conjugate Gradient descent or MR) in a second step. Doing this, the problem of finding local optimums is decreased, while higher computational power is tolerated. In our own work we use Simplex Downhill too. We avoid local optimums by choosing appropriate starting points and by selecting sensor readings from well-located sensor nodes. Thus, computation and storage costs can be kept low.

In subsequent work [20], the nonlinear optimization methods have been replaced by a closed-form linearized least-square solution. A similar approach has been presented earlier in [32]. The closed form solution is much more efficient than the approaches presented before. However, to achieve sufficient accuracy an over-determined system is needed. Wireless sensor network often cannot offer redundant data. In such situations a non-linear solution might produce useful data, while a linearized method fails.

2.4.7 PinPtr - Centralized Sniper Detection

PinPtr [14] presents a centralized sniper detection framework. Nodes distributed in the field are equipped with microphones. The computation of sniper positions is based on time difference of arrival (TDOA) of two different acoustic signals: muzzle blasts and acoustic shock waves. The sensed data is sent to the base station by each node sensing the event. Using this information the base station computes the position of the sniper. A four dimensional consistency function $C_i(x, y, z, t)$ is defined as described in Equation 2.23:

$$C_i(x, y, z, t) = count_{i=1,K,N}(|t_i(x, y, z, t) - t_i| \leq \tau) \tag{2.23}$$

x, y, z describe the hypothetical shooter position and t the shoot time. t_i is the time of arrival of the i th measurement and $t_i(x, y, z, t)$ is the theoretical time of arrival of the muzzle blast at the sensor of the i th measurement. It is defined as described in Equation 2.24:

$$t_i(x, y, z, t) = t + \frac{\sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2}}{v} \tag{2.24}$$

v is the speed of sound and x_i, y_i, z_i are the positions of the node, that has sensed the i-th event. If the i-th measurement is a direct line of sight detection, which means that it is not an echo, then, $t_i(x, y, z, t)$ and t_i should be equal. Due to errors in time synchronization, sensor localization and signal detection the uncertainty value τ is introduced. To find the global maximum of the consistency Equation 2.23, which is the searched location of the sniper, a Generalized Bisection method is applied [16].

PinPtr also supports the self-localization of node positions. This is done by evaluating radio and acoustic signals. The measurements are again transmitted to the base station which computes the relative positions of the nodes.

The localization of node positions and of snipers both require synchronized clocks. This causes additional communication costs. Furthermore, all sensor measurements are sent to the base station. This traffic could be reduced with a distributed approach.

2.4.8 A Statistical Multi-Agent Approach for Event Localization

In [22] the tracking of multiple, interacting targets has been analyzed. In order to estimate the state of multiple events the amount of needed data grows exponentially with the number of targets. Consequently, the communication costs increase too. To solve this, the problem is split into smaller sub problems. The estimation of the event position and the events identity management are handled separately. Location estimation needs frequent local communication as the target has to be tracked continuously. For identity management infrequent long range communication is sufficient, because it is only used if multiple targets are close together and their emission ranges overlap.

If two targets are far away from each other they can be treated as two single events. Coming closer together and finally crossing each others paths the error due to inter-target inference increases. Therefore, the events are estimated jointly. When they separate again, the events will be tracked separately again.

For tracking statistical methods are used. At time instant t the target position $x^{(t)}$ based on the sensor measurement history $z^{(t)} = \{z^{(0)}, ..., z^{(t)}\}$ is estimated. Therefore, a sequential Baysian filtering approach is used. In the presented approach event localization and classification is not mentioned.

2.5 The DELTA Framework

This section discusses the DELTA framework, which supports distributed event localization, tracking and classification presented in [37]. DELTA is described in more detail than the other algorithms as it builds the basis for the enhancements and implementation done in this work. Nearly all functionality is done in a distributed and decentralized way. Only the tuning of the classifiers, which is asynchronously done upon demand, is not done distributed. Nodes are self-organizing and form groups around events and handle them very localized. DELTA does not need to satisfy the condition that the communication range is significantly higher than the sensing range of the nodes. This is achieved by a periodic notification procedure which will be described later in this section. DELTA bridges the gap between the presented cost intensive CSP approaches and the less accurate but more cost efficient approaches focused on minimizing communication load.

2.5.1 The DELTA Protocol

Detecting and tracking moving events by distributed signal processing requires some collaboration among the network nodes. To handle appearing events DELTA establishes dynamic groups and allocates different roles to the member nodes. The state diagram of the algorithm is shown in Figure 2.14. Nodes can be in one of the following five states:

- **LEADER:** The leader node is responsible for group management, localization of the events position and communication with the base station.
- **MEMBER:** These nodes are close to an event and are part of the group which does the tracking and localization of the event. They provide the leaders with the required info.
- **PASSIVE MEMBER:** These nodes are in the neighborhood of the member nodes and are informed about the presence of a leader node. In case of moving targets passive member nodes are prepared for an upcoming event.
- **IDLE:** Nodes which are not in the direct event neighborhood.
- **ELECTION RUNNING:** Nodes in the ELECTION RUNNING state are leader candidates. They have either sensed an event or are members which have lost their leader.

Delta uses the following communication messages:

- **Heartbeat**: Informs the one-hop neighbors, which are the members of a tracking group, about the presence of a leader node. Moreover, they request sensor readings from the members. The position of the sender and the estimated position of the event are transmitted in this message too.
- **Passive Heartbeat**: Informs the passive members about the presence of a leader node. In other words, passive heartbeat messages inform neighbors tow-hops or further away of the leader about the presence of an event. The passive heartbeat dissemination depth can be configured.
- **IREP**: Information response message provide the leader with the needed data. They are also overheard by other member nodes. More about the IREP messages is said in the next paragraph.
- **Leader Reelection**: A leader reelection message is sent by a leader which detects that there exists a member node which is closer to the event than itself. Leader reelection messages are also sent if the signal amplitude of an event sensed at the leader is below a predefined threshold.

As long as there is no event detected the nodes are in state IDLE. Upon starting to sense an event the concerned nodes switch to ELECTION RUNNING state and schedule a timer. The duration of this timer is determined by the sensor readings: the stronger an event is sensed the shorter the timer is set. When the timer expires a heartbeat message is broadcasted. Thus, the neighborhood is informed about the presence of the group leader. Other nodes that are also in ELECTION

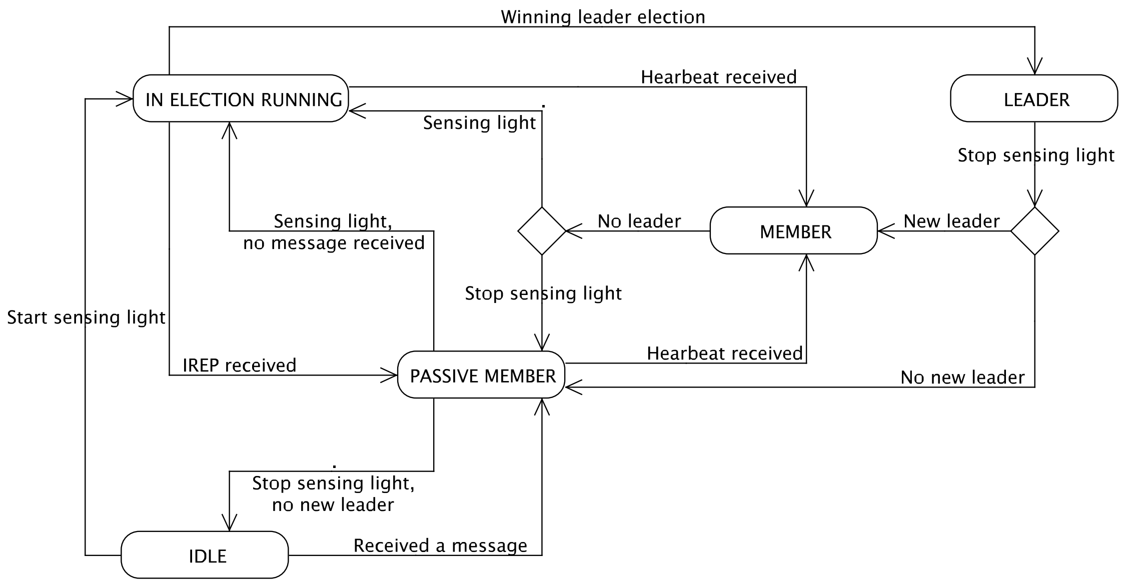


Figure 2.14: State diagram of DELTA.

RUNNING state and which receive this message switch to MEMBER state and cancel their own election timer. The specific computation of the timer depends on the underlying hardware and will be described in Section 3.1.

If events such as a person walking around are moving into the network, a leader handover mechanism is required. DELTA supports this with the following mechanism: Leaders which stop sensing, because the event is out of their sensing range, switch to member state. If member nodes do not overhear a heartbeat message for a certain time and still sense something, they compete for the LEADER state by switching into ELECTION RUNNING state.

As mentioned before, DELTA does not require that the communication range must be larger than the sensing range. If an event with a large emitting range could not only be sensed by a leader and its member nodes, a new, independent group would be built farther away. This would cause the existence of multiple groups for one single event. The information response (IREP) messages prevent this to some extend (see Figure 2.15). Depending on the expected sensing ranges, event information could be distributed deeper into the network by the passive heartbeat mechanism. This implies some overhead, of course. Using optimized broadcasting techniques [15] the overhead could be minimized, though. In Figure 2.15 node D, which is three hops away from the leader could this thus be informed about the existence of the tracking-group.

All packets used in DELTA contain a *round* field. This field is used to avoid the processing of outdated information. It is incremented with every heartbeat the leader broadcasts. If a node receives a packet with a round number smaller or equal than the current round, this message can be ignored. Further, the packets contain a time to live (TTL) field which defines the depth until which the event information is disseminated into the network.

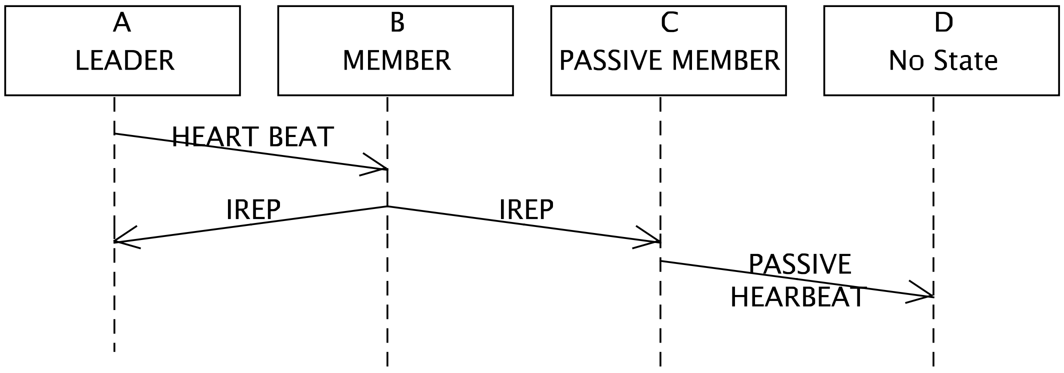


Figure 2.15: Sequence diagram of the DELTA communication scheme.

The leader election process aims at quickly determining a single leader node. This means that the leader choice should minimize the number of reelections and hand-overs, by keeping its LEADER state, as long as possible. The election process needs furthermore to be fast to avoid periods where no leader is present.

2.5.2 Event Tracking

DELTA supports coarse- and fine-grained event tracking. The coarse grained tracking is based on a CPA approach. As described before, leader nodes are the nodes which have sensed the highest amplitude. Leader nodes have a high possibility to be the closest node to the tracked event. Therefore, the event position assumed to be close to the leader node. If the event is moving also the tracking group is moving. So the event can be tracked by tracking the position of the actual leader node. Fine grained tracking, localization and classification of events have not been included into DELTA up to now. These tasks are part of this thesis and will be presented in the next chapter.

2.6 Real-World Hardware Environment

This section is split into three sections. First, we give an overview over the sensor nodes. In the subsequent subsection the operating system running on the sensor nodes is presented. Finally, some details about of the tool chain which was used for developing the DELTA framework on the hardware nodes is given.

2.6.1 Sensor Nodes

For testing real-world scenarios the Embedded Sensor Boards (ESB) [4] (see Figure 2.16) are used. These sensor boards are equipped with several sensor and communication interfaces. The Texas Instruments MSP430F149 [33] is used as microcontroller. It has a CPU clock rate of 8 MHz. The ESB node is powered with three AAA batteries.

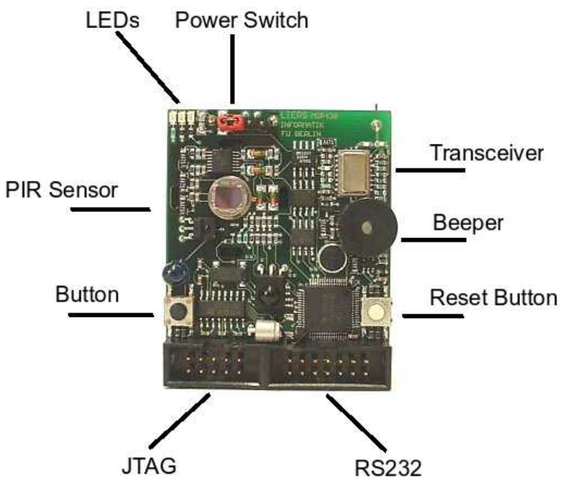


Figure 2.16: The ScatterWeb ESB sensor node.

Sensors

The ESB nodes are equipped with five different sensors:

- **Passive Infrared Sensor**: The passive infrared sensor (PIR) allows the detection of moving objects within a radius of 100°. The maximum measuring distance is, depending on the angle, between 1.5m and 5m [7] [8].

- **Temperature Sensor**: It measures temperatures from -55 °C to +125 °C. The thermometer provides 9-bit temperature readings [5].

- **Microphone**: The ESB boards are equipped with a microphone, which allows detecting noise levels with adjustable thresholds (signal noise ratio 40 dB, 120 dB max) [6].

- **Vibration Sensor**: To monitor tilt and vibrations the ESB contains a vibration sensor [9].

- **Infrared Receiver**: There is an infrared light to frequency converter for receiving infrared (IR) signals on the board which responds to an infrared range from 800nm to 1100nm [10]. In our evaluations this sensor has been used to sense the intensities of the infrared part of light.

Communication Interfaces

The ESB nodes have different possibilities to communicate. On the ESB nodes are three colored LEDs which can be used to signal the state of the node. Furthermore, the nodes are equipped with a parallel interface. This provides a Joint Test Action Group (JTAG) interface over which the nodes can be flashed with new software. Also real-time debugging can be done over the JTAG. Over a serial port the node can exchange data with other devices. For example, the node can sending log data over GSM or via serial cable to a computer. Of course, also the infrared sensor can be used for communication.

The essential communication interface is the radio transceiver TR-1001 [28]. It is ideal for short range communication. Its small size and low energy consumption make it ideal for wireless sensor network applications. The TR-1001 is operating in the license-free 868 MHz band. The signal can either be modulated using On-off Keying (OOK) or Amplitude Shift Keying (ASK). By using OOK a maximal communication speed of 30 kbps is possible. ASK reaches up to 115.2 kbps but consumes slightly more energy.

Microcontroller and Memory

The MSP430 microprocessor is a 16 Bit RISC processor. It supports 27 basic instructions and five different power modes LMP0 to LMP5. Depending on the mode more or less energy is consumed. The modes differ in switching on or off the peripheral modules. The fewer modules are active, the less energy is consumed. Most of the time there is no network administrator around who could reset the system if it crashes. Therefore, the MSP430 is equipped with a watchdog. It increments a timer which has to be reset by the program periodically. If the program crashes or a very time consuming task is performed, the watchdog timer overflows and the system is reset. The JTAG interface on the ESB nodes, allows, as already mentioned, real time debugging on the sensor nodes. On the ESB boards 2 kB of RAM and 64 kB of flash memory (EEPROM) are available. Reading and writing from the EEPROM is energy and time consuming.

2.6.2 ESB ScatterWeb Operating System

ScatterWeb is a tiny operating system which is developed for the ScatterWeb platforms eGate, ECR and ESB, that we use in our real-world experiments. Our implementation is based on version 3.1. ScatterWeb offers the following features [4]:

- Configuration of the sensors.
- Control of peripheral interfaces: Timers, serial port, EEPROM, radio transceiver, infrared communication, switches and LEDs.
- Configuration and management via terminal commands over the serial interface or also over the radio.

Further functionality such as the support of communication over GSM by using SMS can be added upon demand by using the applications. In ScatterWeb the system is split up into operating system and application parts. This means that in most cases a developer does not have to modify the ScatterWeb core code, but can just use the API of ScatterWeb.

When switching on the sensor node the *main()* method is called. Peripheral interfaces are initialized and the main loop gets started (see Listing 2.1). In every loop the watchdog timer is reset. If there is no task to be done, which is signaled by the *runModule* variable, the node is set to sleep state LPM1. In LPM1 state the CPU gets switched off. The CPU can be woken up by interrupts, though. These interrupts set a flag in *runModule* and wake the node up from LPM1 state. The node continues looping and can calls the event handler interrupts. This is needed in our application to schedule events like periodically sending data packets.

```
for(;;) {
    System_startWatchdog();
    // Starts&resets the watchdog, long procedures should call
    // stop watchdog, else MSP will reset.
    if(runModule & MF_SCOS) Threading_eventHandler();
    // Radio tasks.
    if(runModule & MF_RADIO_RX) Net_rxHandler();
    if(runModule & MF_RADIO_TX) Net_txHandler();
    if(runModule & MF_TIMER) Timers_eventHandler();
    // Callback for serial line
    if(runModule & MF_SERIAL_RX) {
      extern volatile UINT8* serial_line;
      if(serial_line != 0) {
       if(callbacks[C_SERIAL]) {
          callbacks[C_SERIAL]((void*)serial_line);
        }
      }
    }
    #if defined(ESB) || defined(ECR)
      if(runModule & MF_SENSORS)
  if(Data_sensorFlags != 0x00) Data_sensorHandler();
    #endif
    #if defined(ESB)
      if(runModule & MF_RC5) Data_RC5ReceiveHandler();
    #endif
    dint();
    nop();
    if(runModule==0) {
      // enter lpm3 and enable GIE at once to not loose an interrupt
      System_stopWatchdog();
      eint();
      LPM1;
    } else eint();
  }
```

Listing 2.1: Main loop in ScatterWeb.System.c.

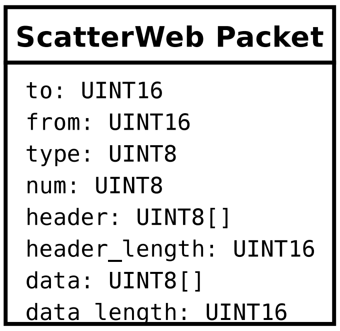
As already mentioned, in time consuming computations it is important to reset the watchdog timer periodically to prevent the system from resetting because the main loop hangs.

Step	**Action**	**Time** at 19.2 kbps	**Time** at 76 kbps
1	Listening to radio	417 μs	105 μs
2	Turning on interface	833 μs	211 μs
3	Sending preamble	3333 μs	842 μs
4	Sending start bytes	1250 μs	315 μs
	Total time	**4583** μs	**1158** μs

Table 2.1: Time needed until a transmission starts using 19.2 or 76 kbps.

Communication aspects

Most of the communication logic is in the file *ScatterWeb.Net.c*. To save energy and avoid packet collisions the transmission power can be set by the user between 0 and 99. ScatterWeb offers a default packet type *packet_t* shown in Figure 2.17. Its size without header and data field is 10 bytes. Most of the fields are self explaining. The *num* field is used to recognize duplicated packets and does not need to be set by the user. The *type* has to be set to identify a packet at receiver side for further processing. A packet can either be sent in unicast or broadcast mode. This is done by setting the *to* field to either a node ID or the broadcast address.



ScatterWeb Packet

```
to: UINT16
from: UINT16
type: UINT8
num: UINT8
header: UINT8[]
header_length: UINT16
data: UINT8[]
data_length: UINT16
```

Figure 2.17: ScatterWeb default data packet.

A packet is sent by calling the *Net_send* routine. The packets get queued in a ring buffer and a checksum is added to the packet. If the buffer is full, new incoming packets are dropped. Before the node starts sending, it checks if the medium is free. If it is not free, a random backoff time is set, after which the node checks the medium again. If the medium is free, the node switches from receive to send state and starts to send a five byte long preamble. The preamble is used to synchronize the radios of the sender and receiver. The three start bytes signal the start of the following data packet. Table 2.1 gives an overview over the time used for transmitting a message with a transmission rate of 19.2 kbps and 76 kbps, respectively. In the example in the table we assume that the medium is free and no other node is sending, so no backoff time is used. Using 19.2 kbps each byte takes 417 μs to be sent. Using 76 kbps it takes 105 μs.

The ScatterWeb implementation uses Manchester encoding. Thus, the amount of data is doubled. At the end of each packet a postamble of 3 bytes is added. Considering unicast packets the sender waits for 30 transmission cycles for an acknowledgment packet. The length of this cycles is depending on the transmission rate. Broadcast packets are not acknowledged. If the sender gets no acknowledgment the node tries up to 15 retransmissions by default.

If a packet arrives at the receiver, the packet is forwarded to the packet handling method which has been registered by the application. The application checks in a *switch* statement the *type* of the packet and decides what shall be done with the packet.

Sensor Aspects

In ScatterWeb each sensor can be enabled or disabled by a system function call. Applications which are interested in sensor data have to register themselves in the callback struct *System_callbacks[C_SENSOR]*. Sensors can be enabled with special parameters such as the interval indicating how often the respective sensor needs to be read. For the DELTA implementation only the light sensor has been used. Details are provided in the next paragraph.

Tool Chain and Development Tools

The development has been done on a Fedora 7 Linux desktop computer. We have used the mspgcc tool chain. This is a port of the GNU C and assembly tool chain to the TI MSP430 microcontroller family. The mspgcc port of GNU C comes with a full set of header files and a basic libC library for the MSP430 microprocessor [34]. Most standard C data types are supported expect double precession floating points. In the package is also a GDB debugger which can be used with the JTAG interface. The GDB debugger can be integrated in multiple graphical front-ends. We have used the Eclipse IDE for coding and debugging.

Chapter 3

DELTA Enhancements

As presented in the previous chapter, DELTA currently supports event detection and tracking. We will extend the framework by adding event localization and classification abilities. Additionally, the CDS backbone mechanism presented in Subsection 2.1 is integrated and functionality which supports updating nodes with new configuration settings is provided. In the first Section we will introduce some changes of the DELTA protocol done in our implementation.

3.1 Adaptations of DELTA Protocol

The implementation of DELTA supporting event tracking, group organization and tracking has been presented in [31]. We introduced some small modifications in the DELTA protocol. In the following section the differences to the reference implementation in [31] are described.

3.1.1 Adaptations of the DELTA Group Organization

Group leaders receiving IREP messages store the information sent by group members for later computations of event positions and amplitudes. In the previous implementation the leader has stopped storing new sensor information, received in the IREP messages, after having collected the required number of sensor values. In our implementation the leader node first stores the minimum required number of sensor readings. If the leader receives more sensor readings, it searches the lowest received sensor reading and replaces it with the new value if the new value is higher . Thus, only the sensor readings from sensors close to the event are stored for further computation.

3.1.2 Definition of Timer Settings

In [36] a leader election timer is mentioned. The ESB nodes sense light values between 0 and 10’000. As nodes with high values should compute short delays and those with low values long delays, the following formula is designed:

$$\triangle t[ms] = \frac{I_{max} - I_C}{10} \tag{3.1}$$

I_{max} is the maximal input value, for light this is 10'000. I_C is the current sensed light value. The leader election delay $\triangle t$ is between 0 ms and 1'000 ms.

3.2 Energy-Efficient Network Management - Backbone Support

The DELTA framework is currently not providing routing functionality and energy efficient node management. It is required that the group leaders of DELTA can report the computed information back to the base station and the network should be running over long time periods. This is enabled by the node management by a CDS. Only the backbone nodes stay awake. Dominated nodes only wake up periodically to update their neighbor table and to transmit pending sensor readings over the backbone. If nodes overhear an event, the DELTA tracking, localization and classification functionality is started. The distributed computed event data will then be sent to the base station over the CDS backbone.

3.2.1 Adaptations of Receiver-Based CDS Protocol

Some small modifications in the presented CDS algorithm (see Section 2.1) have been done. The changes are introduced next.

Learning Phase

The proposed algorithm does not explicitly mention the neighborhood learning phase. Before the CDS setup starts, the nodes broadcast HELLO messages containing their actual state for four hello cycles. This is enough to ensure that every node has overheard a message from its neighbors.

Differences in Dominator Election

In [35] the delay for sending DOMINATOR_CHOICE messages is computed based on the remaining energy of the up-link dominated node, on the nodes receiving the DOMINATED message as shown in Figure 3.1. We moved this decision upwards to the receivers of the DOMINATOR messages. In our implementation these nodes add the delay before sending DOMINATED messages according to their remaining energy level (see Figure 3.2). Dominated nodes with high energy levels send the DOMINATED message faster than nodes with a lower level. In [35] the two-hop neighbors of the dominator receiving the DOMINATED message need to know the energy level of the sender nodes. However, basically they do not need to know the energy level, because the prioritization of the dominated nodes for being elected as dominator node can be done on the dominated nodes themselves. Thus, we save two bytes in the DOMINATED message, because we have not to transmit the energy level of the dominated nodes.

Modification on Message Timers

Figure 3.3 describes the delays and intervals used in the algorithm. We have added an other delay, which is not used in the original paper. To get an overview we introduce all the used delays

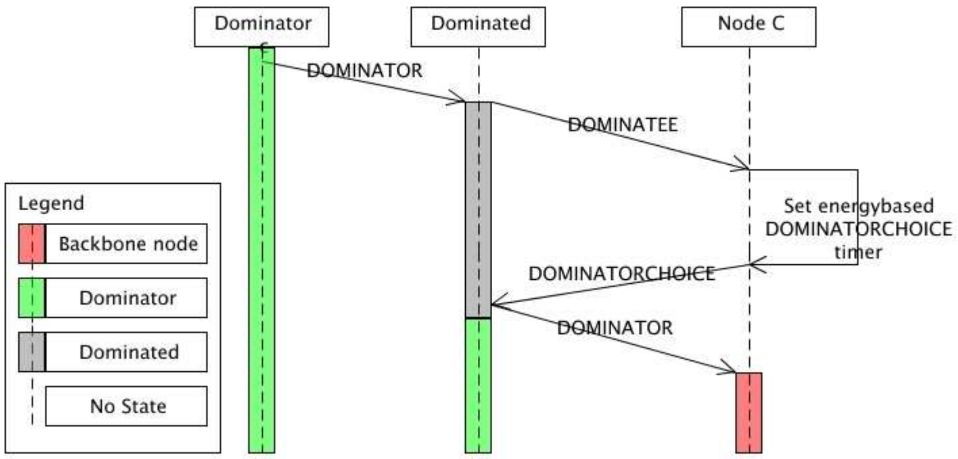


Figure 3.1: Energy based delay set in original version.

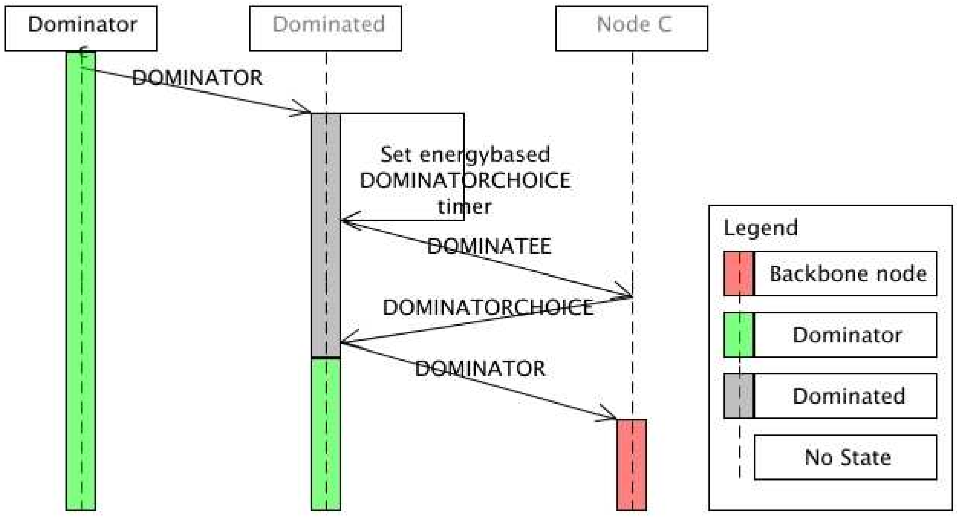


Figure 3.2: Energy based delay set on dominated node.

and intervals. A HELLO message is sent after each HELLO interval with an additional jitter delay if the node is not sleeping. The interval duration is set fixly. The additional jitter is used to prevent packet collisions when sending HELLO messages. It is a relatively long jitter because the possibility of collisions of HELLO messages is higher than for the other packet types. Especially at the beginning when all nodes send HELLO messages. The DOMINATED messages have a shorter jitter. It is again used to avoid collisions. As mentioned before DOMINATED messages are sent with an additional energy delay which is defined by the following function:

$$\text{ENERGY DELAY} = \frac{100 * (\text{MAX VOLTAGE} - \text{MIN VOLTAGE})}{\text{battery} - \text{MIN VOLTAGE} + (\text{MAX VOLTAGE} - MINVOLTAGE)}$$

This function is optimized for the energy levels of the ESB nodes. *MIN VOLTAGE* is set at 2150 and *MAX VOLTAGE* at 2500. Thus, the minimum delay computed by this function is 41 milliseconds while the maximum delay is 100 milliseconds.

DOMINATOR messages are sent with a *SHORT DELAY*. The DOMINATOR_CHOICE message is sent with only a *SHORT DELAY* too. If the nodes have more than one known path to the backbone, an additional delay is added to prioritize DOMINATOR_CHOICE messages of nodes with only one known path to the backbone.
The length of the timer for *LINK BREAK* messages is at least as long as a *VERY SHORT DELAY*, but not longer than a *SHORT DELAY*. From this range the length is randomly chosen. The length of the *LINK REPAIR* timer is at maximum equal to the length of a *VERY SHORT DELAY*. We have introduced the restrictions on the *LINK BREAK* messages and the *VERY SHORT DELAY* because the *LINK REPAIR* message has to be sent before a *LINK BREAK* message. This is necessary to prevent a distribution of the link break in the whole network. The link break should be repaired as local as possible by *LINK REPAIR* messages. The absolute values of the timer lengths are discussed in Subsection 4.2.1 where the other implementation details are discussed.

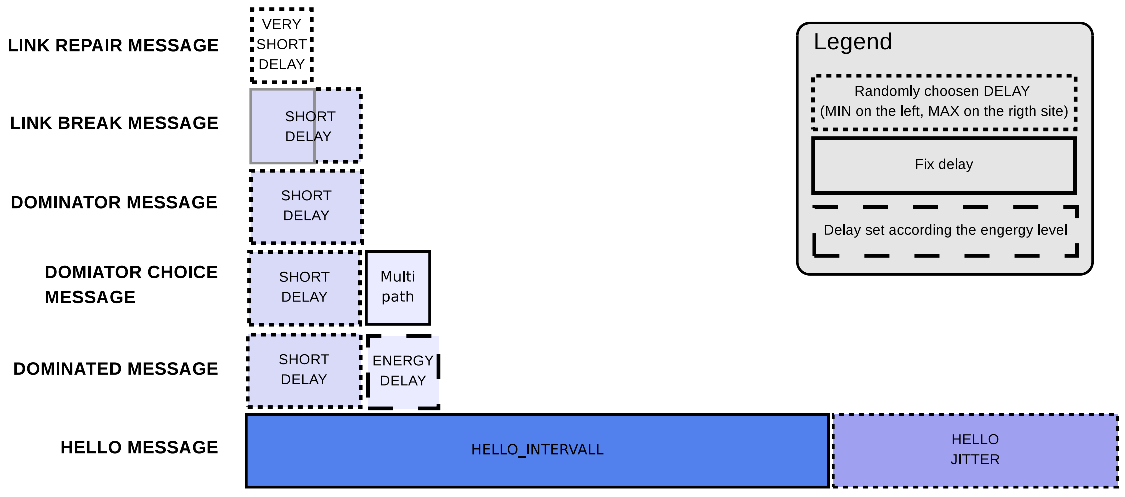


Figure 3.3: Delays for each packet type.

Optimizations for non Bidirectional Links

In real-world the links between sensor nodes are not always bidirectional and the link quality is varying. Therefore, we have introduced some filters which were not foreseen in [35]. When receiving a DOMINATED message it is not only checked if the receiver node has any links to the dominator, but also if the dominator is not already a direct neighbor of the receiving node. This could be possible because the intermediate dominated node has not overheard a DOMINATOR message due to temporal link problems.

DOMINATOR, DOMINATED and DOMINATOR_CHOICE messages are all sent not only once but for a predefined number of times. The number of retransmissions is set to five. The first message is sent after a normal delay (see Figure 3.3). The following messages are sent with additional retransmission delays to prevent an overloading of the network. Because we use retransmissions, there is an additional state check necessary: The nodes have to register if they have already received a DOMINATOR message from a certain node to prevent handling a message twice.

Changes in Link Repair Mechanism

There are also some small changes in the repair algorithm. LINK BREAK messages are enhanced with a Time To Live (TTL) field. This helps to keep link-breaks locally. Furthermore, in our version we handle only one LINK BREAK message at the same time. We think that it does not make sense to handle more than one. Very often LINK BREAKs have the same origin (i.e. the same failed dominator). In [35] nodes which overhear a LINK REPAIR message from a node which is in their list of broken nodes cancel their own link update procedure. We changed this slightly. All nodes which started reporting a link break and receiving a LINK REPAIR message just set their state to dominated immediately and do not forward the LINK REPAIR message further. This limits the number of resulting dominator nodes after the repair process.

3.2.2 Definition of the CDS Data Packets

[35] does not provide any details about the design of the communication packets. As mentioned before in Section 2.1.4 the following data packets are foreseen: HELLO, DOMINATOR, DOMINATED, LINK BREAK and LINK REPAIR messages. In Figure 3.4 all packets sizes and the needed fields for each message type are shown.

All packet structures in Figure 3.4 are embedded into the field $data$ of the default ScatterWeb data packet shown in Figure 2.17. The HELLO packet has a field called *nodeOrConfigID*. This field is used by the configuration updated mechanism and indicates new configuration updates that are available. In the dominator packet is a field called *dynData*. This array contains the neighbor table and the source path. In our configuration the neighborhood table is an array with maximal eight neighbors and the path is an array of the node IDs with a maximal length of 5 hops. We take these two arrays into one because of technical reasons: Often the neighbor table or the source path are not completely occupied. As we do not want to send empty arrays we only send the filled fields of these arrays. Therefore, these two arrays have a dynamic size. If

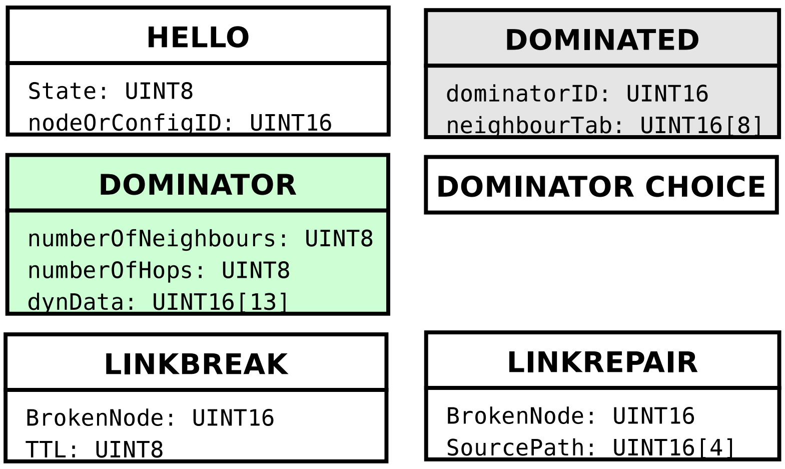


Figure 3.4: Data packets of CDS.

there were two different variables with dynamic size in the data packet, the receiver would not know at which memory address it should start reading for the second array. In other words the pointer for the second field would not be at the expected position. With only one dynamic field the receiver just has to know the sizes of the first and the second array. Thus, it can first read from the start address of *dynData* to the end of the first array and then from that address to the end of the second array.

As you can see, the DOMINATOR_CHOICE packet has no additional data fields. The receiver ID is the only value which has to be sent and is included in ScatterWebs default data packet.

3.3 Energy-Based Source Localization

Up to now DELTA provides no efficient localization algorithm. As the algorithm needs to be implemented on hardware with low processing power and with memory restrictions, the range of adequate localization algorithms based on sensed energy levels is limited. We have been looking for an efficient and reliable algorithm which computes event positions and amplitudes using the sensor measurements according to the energy-based decay model described in Equation (2.2). We present three different algorithms (see Section 2.4) and evaluate them in Section 5.3. These are Linear Least Square, Simplex Downhill and Conjugate Gradient Descend. Simplex Downhill and Conjugate Gradient Descend are nonlinear optimization methods which again minimize the square error. The algorithm with the best performance will be integrated in DELTA for a appropriate event localization.

The sensor model on which all three presented algorithms are based is defined in Equation (2.2). The unknowns are the coordinates of event $\mathbf{x}$ and its emitted amplitude c. If the event has to be detected in space, $\mathbf{x}$ has 3 dimensions. So to solve the according system we need at least measurements from five sensor nodes. This implies that the sensor readings of at least 5 (4 members and + 1 leader) sensor nodes are required. If the system is solved with Linear Least Square, the accuracy might not be sufficient. An over-determined system is needed to improve accuracy of Linear Least Square. The three localization estimation algorithms Linear Least Square, Conjugate Gradient Descend and Simplex Downhill, presented in the previous chapter are evaluated in Section .

3.4 Event Classification

The estimated localization information and event amplitudes computed on a sensor node, by either Linear Least Square, Simplex Downhill or Conjugate Gradient Descend, can be used for the classification of sensed events. Classification can be interesting for different tasks. In some applications it is not sufficient to know only the position of an event but, also the kind of event needs to be known. This might for example be the case when classifying a vehicle (bicycle, car, lorry). If the classification of the events is done on the sensor nodes, less information has to be routed to the base station, because not all the sensed data has be sent to the base station. One can also do dedicated filtering on the sensor nodes: If the base station is only interested in some kind of events, only information about these events will be routed to the base station.

For finding the clusters we will introduce K-Means (see Section 2.4) into DELTA. It is an unsupervised algorithm which is used to identify clusters in existing data sets. These clusters represent the different event types (classes). The learning phase is performed at the base station. In a learning phase different event types are fed to the system. The sensor nodes compute emitted signal strength and position with DELTA. This information is routed to the base station, where it is stored as training data. The clusters are then extracted from this training data by K-Means. Then the cluster information is distributed to the sensor nodes, where it is used for event classification.

The Bayes classifier allows us to classify n-dimensional events. This could for example be an event with light and sound emissions. To apply the Bayes classifier on the sensor nodes the following parameters have to be precomputed at the base station:

- The covariance matrix $\mathbf{K}_i$ for each class C_i.
- The mean vector $\mathbf{m}_i$ for each class C_i.
- The probability $p(C_i)$ for each class C_i.

These three parameters can easily be determined from the K-means output and are distributed and applied onto the sensor nodes with the protocol presented in the next section.

For the classification the Bayes classifier and the Minimal Distance classifier will be implemented an evaluated.

3.5 Configuration Distribution over the CDS

Once the sensor nodes are deployed, it still should be possible to change their configuration settings. The tasks of DELTA nodes remain more or less the same. The nodes have to classify events. So it does not make sense to implement a complex update functionality, which supports the replacement of the whole behavior of a node by exchanging the application software on the nodes. Simple adjustments in the sensor network are done by only changing some configuration parameters. A concrete example: The delta nodes are able to classify events by using a classifier. Using Bayes classifier, the covariance matrix, the mean vectors, and the probabilities for each class have to be updated on the nodes if the event classes change (see Subsection 2.4.5).

To update such configuration settings we have implemented a simple protocol which supports dissemination of configuration information in the network. The protocol is based on the receiver-based backbone algorithm presented in section 2.1. The backbone is used to distribute the information. In the actual real-world implementation the algorithm is used to distribute the classifier settings for Bayes classifier. This algorithm could also be used to distribute other configuration settings to the sensor nodes.

The distribution algorithm is designed to keep the network traffic low. The control messages of CDS are used to signalize new configuration settings. No new control packet is introduced. The algorithm is designed to reach as many neighbor nodes as possible with one single update packet. The node which sends the update packet waits until all its neighbor nodes are awake and ready to receive the update packet.

Each configuration setting has its own version identifier (VID). The VID is incremented with each update. A new field is added to the HELLO packet of the backbone construction algorithm to signalize the version of the actual configuration setting. Each node adds the VID to the HELLO packet. So the HELLO packet size has one additional byte.

Initially, the network administrator updates the configuration setting at the base station. The VID of the base station is incremented by one. The incremented VID is broadcasted with each HELLO message. The nodes receiving the HELLO packet check if their VID is smaller than the broadcasted VID. If this is the case, the node sends a GET packet with the desired VID to the base station. The base station sets a timer to broadcast an update packet. The length of this timer is set to one sleep interval. This timer is used to guarantee that all neighbors overhear the update message which contains the updated configuration settings. Dominated nodes periodically sleep for one sleep interval. After that, they overhear HELLO message from their dominator. If this HELLO message contains a higher VID as their own, the dominated node also sends a GET request and stay awake until it receives the update message. Thus, all neighbors can be updated with one single update message.

Figure 3.5 describes the whole update process. The base station sends HELLO messages which indicate a new configuration setting. The Dominator node overhears this message and sends a GET message to the base station. The base station sets the timer to send the update packet. The dominated node is sleeping and has not overheard the HELLO message yet. After the dominated node wakes up, it overhears the HELLO message which indicates the new configuration settings. It also sends a GET message to the base station and suppresses its sleep timer. Both, the dominator and the dominated node get the update packet and update their configuration. The dominated node goes back to sleep again after successful reception.

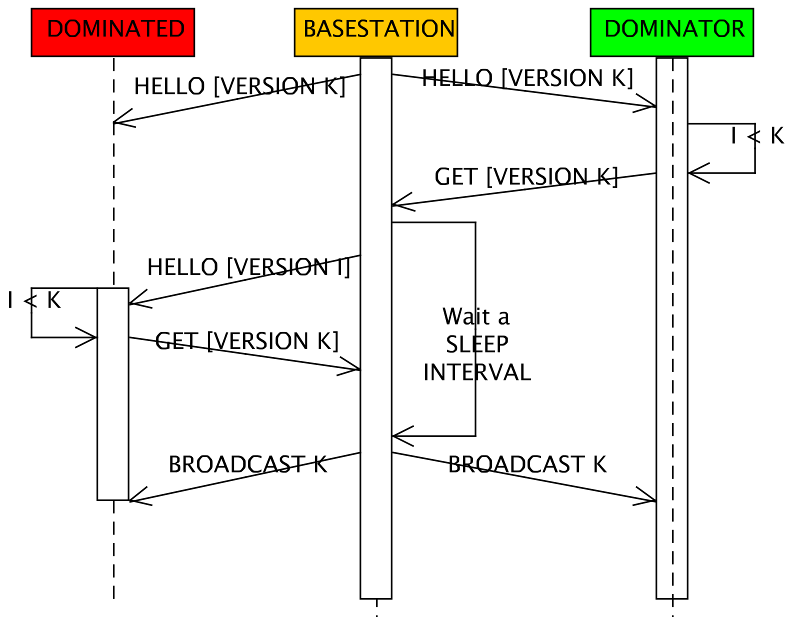


Figure 3.5: Sequence diagram of configuration distribution protocol.

Chapter 4

Implementation

pThe DELTA framework with its extensions have been implemented on real-world hardware. We first present the modification of the operating system of the used hardware, which is used to evaluate the algorithms. After that we present some more details about the implementation of DELTA and the CDS backbone.

4.1 Modification in the ScatterWeb OS

We tried to keep modifications in the ScatterWeb firmware code at a minimum. Whenever possible our code has been implemented in the application layer. There are some cases where this was not possible.

For the testing of DELTA we have used the light sensor. The light measurements done by the ScatterWeb software are not appropriate for our purpose. The original implementation was only able to make binary decisions (light on / off). So, we had to modify the implementation of the light sensor readings. When the light sensor is switched on the current light value is read periodically. A mean value and the current light value are saved. This value is used as input for the leader election process.
Some modifications are also done to adapt the communication behavior. When the radio is switched on the ring buffer is cleaned by default. Dominated nodes in the CDS are switched off most of the time (see Section 2.1). In the ScatterWeb implementation the application has to check if the radio is switched on before any transmission is started. If this is not done, data could be lost. If the radio is switched off and data is sent, it is written into the ringbuffer. By default, ScatterWeb deletes the whole ringbuffer when switching on the radio. We changed this behavior: When switching on the radio, we check first if there is something in the ring buffer and start sending possible data at start up automatically.

We have removed the Manchester encoding and have replaced it by applying an XOR operation on the data with 0xAA. Thus, bandwidth usage is reduced and the sending of long 0 bit series is prevented. Finally, we have changed the bandwidth settings in the firmware from 19.2 kbps to 76 kbps.

We have encountered a bug in the macro, which checks if there is some space left in the radio ring buffer, which was responsible for buffer overflows. The bug has been fixed in ScatterWeb version 3.2.

Implementing the whole DELTA framework on the ESB nodes, the nodes would run out of memory. Therefore, we removed all the methods and variables from the firmware which are not used in our application.

4.2 CDS Backbone

As mentioned in section 2.1.4 we have chosen receiver based CDS for real-world implementation. In the following section we present the real-world implementation details of the backbone algorithm.

4.2.1 Details on the Implementation of the CDS Algorithm

In this subsection we introduce the details of the implementation of the CDS algorithm. The timer-based CDS protocol is described in Section 2.1.4, its adaptations are described in Section 3.2.1. First, we give an overview of the state variables used in the CDS algorithm. In Table 4.1 the first row shows the names of the state variables. The second row shows the memory size of each variable and the third row gives a short description of each state variable. The sizes of $neighbour\_tab$ and $neighbour\_detail\_tab$ have to be set before deployment. It is depending on the expected number of neighbours. The two tables store information about the nodes one-hop neighbors. The neighbour table is splitted into two variables. The *neighbour_tab* is sent within the dominator messages, where as the *neighbour_detail_tab* is only used locally on the node. The three counter variables $nrOfDominateeMesSent$, $nrOfDominatorMesSent$ and $nrOfDomChioceMesSent$ have a size of two Bytes, where as the $repairMesCounter$ is only one Byte long. This is because the first three counter variables are passed as parameters to the timer setting function. The function only supports parameters with a size of 2 Bytes.

We already gave an introduction into the functionality of the algorithm in Section 2.1.4. In the following paragraph we describe in detail when and how nodes change their state. For each incoming control message type we present a code listing. Pseudocode listing 2 describes the handling of an incoming DOMINATOR message. Incoming DOMINATOR messages are only handled if the senders ID is not the one of the actual dominator node of the receiving node. Dominator messages from the same node are handled only once. Pseudocode 3 handles DOMINATED messages. Incoming messages are only handled if the receiving node has not yet set a dominator choice timer. If the receiving node has more than one path to the dominator node an additionally delay is set. Pseudocode 4 handles DOMINATOR_CHOICE messages. DOMINATOR_CHOICE messages are only handled, if the message is addressed only to the receiving node. Pseudocode 5 handles LINK_BREAK messages. Messages are only handled if the receiving node is not already in link break state or has scheduled a link repair message. Dominated nodes forward the LINK_BREAK message and dominator nodes set a link repair timer. Finally, Algorithm 6 handles the LINK_REPAIR messages. Only nodes in link break state handle these messages. Nodes which have already received this LINK_REPAIR message remove their link repair timer and set their state to dominated.

Name	Size [Bytes]	Description
$nodeState$	1	Describes the state which is set to the actual node color.
$myDominator$	2	ID of the nodes dominator node.
$myDominatee$	2	ID of the node which has sent a DOMINATED message. Used in the network construction process.
$domChoiceTimerSet$	1	Indicates if the DOMINATOR_CHOICE timer is set or not.
$isSleeping$	1	Indicates if node is in sleep mode or not.
$nrOfDominateeMesSent$	2	Indicates the number of sent DOMINATED messages.
$nrOfDominatorMesSent$	2	Indicates the number of sent DOMINATOR messages.
$nrOfDomChioceMesSent$	2	Indicates the number of sent DOMINATOR_CHOICE messages.
$repairMesCounter$	1	Indicates the number of sent LINK REPAIR messages.
$neighbour\_tab$	16	IDs of the neighbor nodes. (8 neighbors at maximum).
$neighbour\_detail\_tab$	40	Time stamp of the last received message and state of the neighbor nodes. (8 neighbors at maximum).
$numberOfNeighbours$	1	Actual number of neighbors.

Table 4.1: State variables of each node.

```
Input: DOMINATOR message
Update neighbor table;
if packet sender != myDominator AND (nodeState = uncovered OR nodeState =
inelection) then
    myDominator ← sender ID;
    nodeState ← inelection;
    if I have uncolored neighbors then
        compute energy based delay;
        set DOMINATED packet timer;
    else
        nodeState ← dominated
```
Pseudocode 2: Incoming DOMINATOR message.

```
Input: DOMINATED message
if domChoiceTimerSet = 0 AND (nodeState = inelection OR nodeState = uncovered) then
    Update neighbor table;
    myDominatee ← sender ID;
    if I am neighbor of the DOMINATOR node OR there is more than one path to the
    dominator node then
        set DOMINATOR_CHOICE packet timer with additional delay;
    else
        set DOMINATOR_CHOICE timer without additional delay;
```
Pseudocode 3: Incoming DOMINATED message.

```
Input: DOMINATOR_CHOICE message
if Message is for me AND nodeState = inelection then
    nodeState ← dominator;
    set DOMINATOR timer;
```
Pseudocode 4: Incoming DOMINATOR_CHOICE message.

Next, we give an overview over the settings of the intervals and delays. The setting of the timer lengths has influence on the network lifetime. Network topology and the node mobility have to be considered when timer lengths are set. The longer the hello interval is set, the longer the dominated nodes have to listen until they overhear the first HELLO message. This means that the listen periods have to be set longer, so that dominated nodes receive hello messages from their dominator nodes. Moreover, the backbone setup process takes longer with long hello interval because the neighbor learning phase is longer. On the other hand, the dominator nodes save energy. Dominator are sending HELLO messages constantly. The longer a hello interval is, the less HELLO messages are sent. The length of the sleeping period is also influenced by the required frequency to report sensor readings to the BS. If sensor readings have to be

```
Input: LINK BREAK message
if LRMesTimerSet not set AND not already in LINK BREAK state AND (nodeState =
dominator OR nodeState = dominated) then
    decrement TTL of link break message ;
    if nodeState = dominated then
        nodeState ← linkbreak;
        if broken node ID of link break message is my own ID) then
            Set link break timer using the ID of the sender for the broken node ID;
            /* The sending node has not overheard link repair message of my LINK
            BREAK. My link is already repaired. So froward the message with the
            senders ID. It would also be possible to send a LINK REPAIR message back
            to the sender.*/
        else
            Set link break timer;
    if nodeState = dominator then
        if Is link break origin in my path then
            Send REPAIR message with a delay;
        else
            Send REPAIR message immediately;
```

Pseudocode 5: Incoming LINK BREAK message.

```
Input: LINK REPAIR message
if nodeState = linkbreak then
    update neighbor table;
    update source path ;
    if I am the origin of the link break message then
        nodeState ← dominated;
    else
        if ! LRMesTimerSet then
            set repair timer;
        if LRMesTimerSet AND an other node was faster with forwarding this message
        then
            remove repair timer;
            nodeState ← dominated ;
```

Pseudocode 6: Incoming LINK REPAIR message.

reported often, the sleeping period has to be set shorter. If we have a dynamic network topology, it may be necessary to set shorter long intervals. Moving nodes cause a lot of link breaks. This leads to a more and more suboptimal backbone. Shorter reestablishment cycles would

Interval / Timer	length [s]
HELLO	6.6
LISTEN PERIOD	13.1
SLEEPING	65.5
LONG	1769.5

Table 4.2: Length of the timers.

rebuild the whole network periodically. With the timer configuration presented in Table 4.2 we maximize the lifetime of the network. The settings are for a static network. The sleeping time of the dominated nodes is maximized by long sleeping intervals and short listen periods. The maximal timer length of ScatterWebs timer library is limited to 65.5 s. We set the length of the sleeping interval to 65.5 s. Based on the sleeping interval we computed the other intervals by multiplication or division. The listen period is five times shorter than the sleeping interval, the hello phase is ten times shorter, and the long interval 27 times longer than the sleeping interval, which is around half an hour. The listen period has to be at least as long as the hello interval. We have set the listen period twice as long as the hello interval to reduce link breaks because the possibility to overhear a HELLO message is higher.

4.2.2 Data Traffic

This subsection describes how data messages are routed over the backbone. Data packets are sent using the standard ScatterWeb packet type. Meta information about the data packet is set in the header fields of the ScatterWeb packet as illustrated in Figure 4.1. At the base station functions are registered to handle the different data packets. This is done by the function *CDS_registerDataHanler(UINT8 messageType, dataHandlerP_t fp)*, where *messageType* is the type of the data packet of interest and *fp* is a function pointer to which the packet has to be sent for further processing. The registration is done dynamically at run-time.

To send data to the base station the function *CDS_sendToBase* has to be called. Because each node in the backbone knows its dominator node, a data packet is simply routed along the backbone from one node to the other. Each hop reads the meta information in the header field and increments the hop counter. If a packet arrives at the BS, the information transmitted in the data field of the ScatterWeb packet is forwarded to the function which is registered to process the data.

4.3 Localization Using Simplex Downhill

In section 3.3 the localization algorithms Linear Least Square, Simplex Downhill and Conjugate Gradient Descend are described. In evaluations (see Section 5.3) Simplex Downhill has shown best performance in simulator tests. Therefore, only Simplex Downhill has been implemented on the ESB hardware for real-world tests.

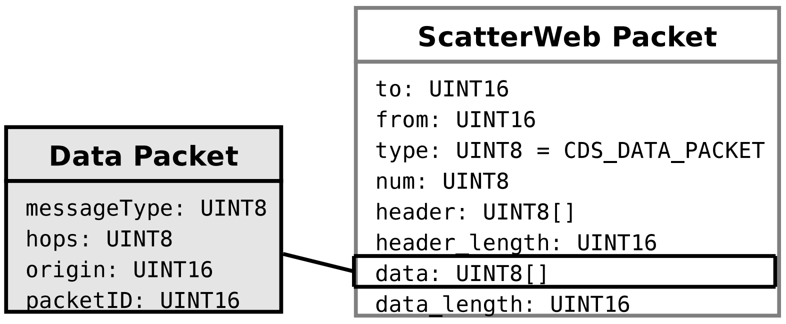


Figure 4.1: Generic data packet used by the CDS approach.

Simplex Downhill is implemented in two versions: In the first implementation, Simplex Downhill computes the 2-dimensional position (x and y) and the amplitude of an event. In a second implementation Simplex Downhill only computes the events position X (x, y) without computing the event amplitude. Having computed X, the amplitude c of an event can be computed using the following equation:

$$c = ||X - \xi_i||^2 * \rho_i \tag{4.1}$$

Where X is the leader position and ρ is the sensor reading of the leader. In other words, by multiplying the square distance between the event and the leader with the sensed amplitude, we get the intensity amplitude of the event.

The start points of the Simplex Downhill computation influences the result. We define the point located at the center of area of the sensing nodes and their measurements as the starting point for the Simplex Downhill algorithm. This point is likely close to the sensed event.

Simplex Downhill has different configuration parameters:

- **DIM**: Number of dimensions of the problem. Normally the dimension is three (x, y, z).

- **NMAX**: This is one of the two termination conditions. It gives the maximal number of iterations. If the maximum is reached Simplex Downhill terminates.

- **FTOL**: Is the second termination condition. If the distance between the highest and the lowest value of simplex is smaller than FTOL, the algorithm terminates.

- **NO_OF_REF_POINTS**: The number of reference points which are required in the Simplex Downhill computation.

- **COMPUTE_WITH_AMPLITUDE**: A boolean variable which defines if the amplitude should be computed with Simplex Downhill or afterwards.

4.4 Event Classification

The presented classifier and clustering algorithm in Section 3.4 have both been implemented in a simulator and on the ESB nodes.

4.4.1 Implementation in Simulator

To get an impression of the performance of the classifier algorithms Bayes classifier and minimal distance classifier and of the clustering algorithm K-Means, all mechanisms have been implemented in MATLAB. MATLAB offers a classifier toolkit which contains the K-Means clustering algorithm [24]. The K-Means implementation of MATLAB minimizes the *sqEuclidean* distances for each sample to the cluster center by default. We have the default settings in our simulation. The K-Means implementation allows supplying initial cluster centers.
The implementation of Bayes classifier is simple in MATLAB. Only function 2.22 described in subsection 2.4.5 has to be implemented. An input value is applied to each cluster to get the membership values. To compare the performance of minimal distance classifier and Bayes classifier the number of wrong classified elements is compared.

4.4.2 Implementation on Sensor Boards

On the sensor boards we have implemented Bayes classifier for one-dimensional classification only. It is the same function as used in the simulator environment but supports only one-dimensional input. We have used only light sensing inputs for event classification on the sensor boards.

The Bayes classifier implementation uses three arrays of type *float*. One array keeps the probabilities of the classes, one the means and one the covariances. The lengths of these arrays depend on the number of classes which have to be classified.

4.4.3 Bayes Classification Configuration Distribution

The classification with Bayes classifier depends on the three arrays described before. These arrays are learned at the base station by applying the K-Means algorithm and have to be distributed to the network nodes afterwards. The event characteristics can change during runtime or new event types can evolve. This means that also the classification parameters have to be adapted. This is done via the update mechanism described in Section 3.5. In Figure 4.2 the data packet for configuration updates is shown. If a sensor node receives a packet with a higher *configID* than the currently stored one, the Bayes classifier configuration arrays are overwritten on the node with the received information.

4.5 Logging Environment

For the evaluation and the testing of the different implementations it is crucial to have appropriate logging data. Therefore, we have implemented a logging environment which supports the handling of log information generically. Three different ways of logging are supported:

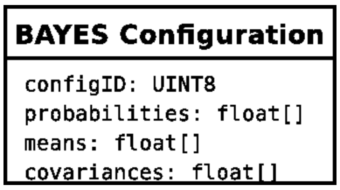


Figure 4.2: Packet for Bayes Classifier Update.

- **Console**: Each sensor writes logging information to the console. This information is captured over a serial cable for further processing.
- **Memory**: The logging information is written into the EEPROM of the nodes. This information can later be read over a serial cable for office analyses. As the ESB nodes have very limited memory, the information which can be saved is limited.
- **Radio**: The logging data is sent over the radio to the BS. This information is sent directly and does not use the CDS data packet routing. For most testing scenarios this method is not appropriate as the data sent over the radio falsifies results.

The logging messages are standardized. A log message consists of three fields as illustrated in Figure 4.3. The first field is the type of the log message. The next two fields are parameters,

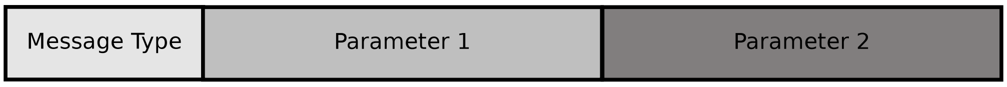


Figure 4.3: Logging data structure.

which are two UINT16 fields. In these fields the log details can be written. It is also possible to use only one or none of the parameters. An example: A data packet arrives at the BS. The log entry would be [DATAPACKET_IN, *packetID*, *sourceID*], where the type of DATAPACKET_IN is a UINT8. Depending on the log configuration these three fields are either written into the memory, sent to a predefined logging node which prints out the logging information, or are printed directly to the serial interface. In the last two cases the information is made human readable. In our example the output would look something like ”-IN-;DATA PACKET;102;0202”. This information is then processed into a log file or a database.

The size of a single log entry is 4 bytes. According to the ScatterWeb documentation the EEPROM size is 64 kilo bit. So, maximal 2000 log entries could be saved on the ESB nodes. The configuration of the logging settings can be done over the radio at run time.

Chapter 5

Evaluation

In this chapter the CDS algorithm and the DELTA framework with its extensions are evaluated. To save energy and to support multihop communication, the sending power has been adjusted. We present the evaluation and the results of the CDS backbone tests. Finally we evaluate the tracking, localization and classification of DELTA and discuss the results.

5.1 Adjusting the Transmission Power

As mentioned in Subsection 2.6.2 the sending power can be set between 0 and 99 in ScatterWeb. We are interested in an optimal ratio between sending power and node distances for multiple reasons: Sending with full power is energy consuming. So we tried to minimize the sending power in order to save energy. The distances between two nodes should not be too large, because, depending on the event which should be localized, sensor nodes are densely deployed. Nodes should be overheard only by their physical neighbors. Communication among nodes which are two hops away should be improbable. This reduces packet collisions and saves energy.

The network topology should support transmission ranges of approximately two meters. On point 0 the source node S is placed. We made two test setups. In the first one 4 nodes are placed at a distance of 75 cm and 4 at a distance of 150 cm from the sending node S. For the second setup 4 nodes have been placed at 125 cm and 4 at 250 cm. We have used 4 nodes at each distance to crosscheck the nodes receivers. The test bed settings are depicted in Figure 5.1.

Node S is sending with a sending power of 10, 12, 14, 16 and 18. For each sending power 100 data packets with a size of 35 bytes have been sent. This test was repeated 4 times with different sending nodes. So, each node should receive ideally 400 data packets for each tested sending power value.

Figures 5.2 and 5.3 show the average number of packets which have arrived with the different sending powers. Comparing Figures 5.2 (a) and (b) with Figures 5.3 (a) and (b) we see that the distribution in the first two is much lager. In Figure 5.2 there is no sending power setting, where packets are reliably received by nodes at 75 cm but not by nodes at 150 cm.

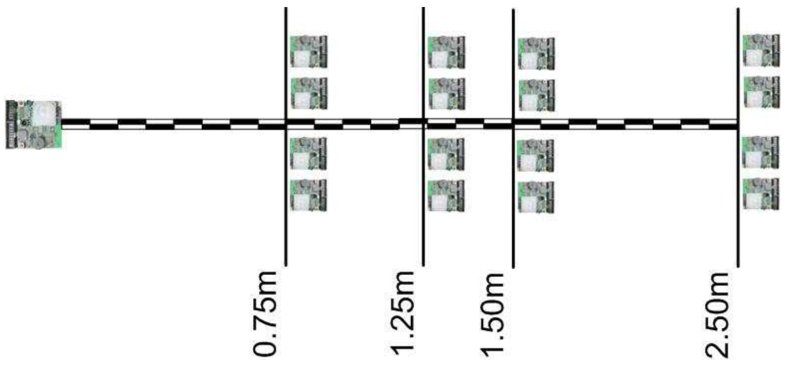
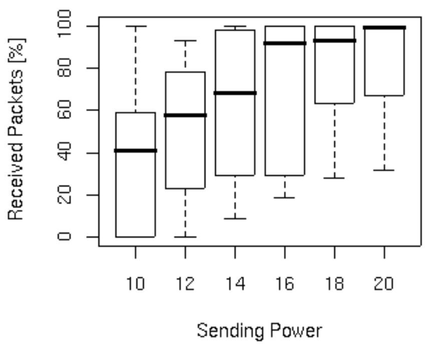
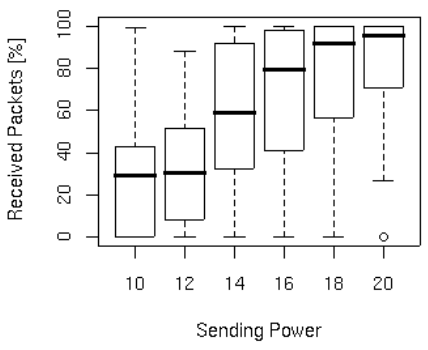


Figure 5.1: Test bed for determining the transmission power.

Comparing Figures 5.3 we see that with a sending power of 16 most packets are overheard by nodes at 125 cm (median of 85%) but only few at 250 cm (median of nearly 0%). So, a sending power of 16 for node-to-node distances of 125 cm is appropriate.



(a)

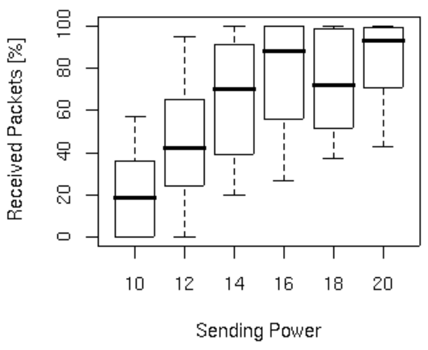


(b)

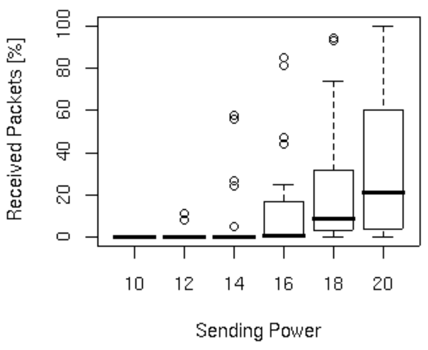
Figure 5.2: Packets arrive at 75 cm (a) and packets arrive at 150 cm (b).

5.2 Testing the CDS Algorithm

For testing the CDS two different test cases have used. In the first case, basic functionality such as proper backbone construction and if the repair algorithm works as expected is tested. In a second scenario we run the CDS network over a longer period to analyze its long-term performance.



(a)



(b)

Figure 5.3: Packets arrive at 125 cm (a) and packets arrive at 250 cm (b).

Interval / Timer	length [s]
HELLO	2
LISTEN PERIOD	6
SLEEPING	10
LONG	65.5

Table 5.1: Length of the timers for the CDS experiments.

5.2.1 Basic Functionality of CDS

We have already mentioned the settings of the timers and intervals for the CDS. For the following evaluation we use the timer configuration listed in Table 5.1. The intervals are shorter than the ones described in Table 4.2 to get shorter test cycles. Changing the intervals does not basically change the behavior of the backbone algorithm. The tests are started by sending an initial signal over the serial cable to the base station. Then the tests run for one LONG cycle. After that all nodes are connected to the serial interface. The data which the nodes have written into their EEPROM is read and written into the database.

Backbone Setup

The network topology in the first experiment is a simple grid as depicted in Figure 5.4. We check in this experiment if the backbone is well established and how long it takes until the path setup is finished. The experiment has been repeated 10 times. To simulate network traffic each node tries to send a four byte data packet every 10 seconds as soon as they are in dominator or dominated state.

The path setup times we have got in the test runs are between 14.8 seconds (s) and 17.0 s . In average it was 15.6 s with a standard deviation of 0.7. The setup process starts when the first HELLO packet is sent to learn the neighbor tables. The build process ends, when the last DOMINATED or DOMINATOR packet is sent. Before the CDS algorithm is started, four HELLO messages are sent to build up the neighbor tables. With the settings of Table 5.1 this makes $4 * 2s = 8s$. The nodes which are sending DOMINATED or DOMINATOR messages send the first message with a short delay of at maximum 0.2 s. After that they send four more messages with an additional retransmission delay of 1 s. Thus, we get another $4 * (1s + 0.2s) + 0.2s = 5s$. The learning phase of 8 s plus the time it takes until the last node stops sending DOMINATOR or DOMINATED messages of 5 s results in 13 s. The remaining 2.6 s are used for the effective network build process. The average duration of 15.6 s used in the experiments is therefore feasible. The optimal path is depicted in Figure 5.5. The green nodes represent

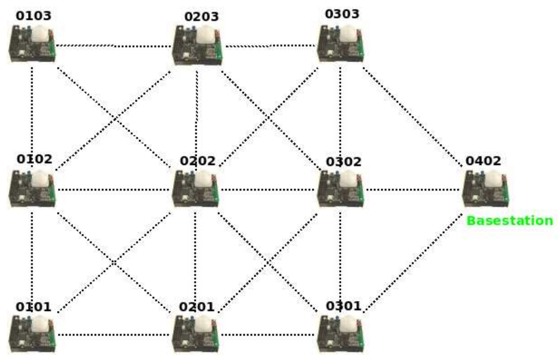


Figure 5.4: Test case grid.

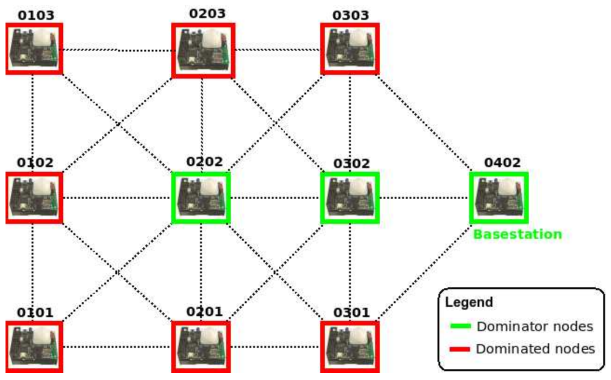


Figure 5.5: Test case grid with optimal path.

dominator nodes in the CDS, the red ones are dominated nodes. In average in all 10 runs 3.4 dominator nodes have been elected. As shown in Figure 5.6, there was a case where five nodes have been chosen. We investigated this case in detail. The five elected dominator nodes are the base station (0402) and the nodes 0302, 0303, 0201 and 0202. By analyzing the network traffic in this case we found a lot of lost packets.

Figures 5.7 (a), (b) and (c) illustrate the control message traffic during the net setup phase for the experiment mentioned above. In all subfigures the CDS backbone nodes of the final CDS are shown. Only the messages which arrived at the receiver nodes are depicted. Node 201 receives the DOMINATED message from node 302. Nodes 202 and 203 got their DOMINATED message from node 303. So node 303 was elected by node 203 and node 302 by node 201. At this point already 3 dominators are elected. Then nodes 302 and 303 sent their dominator message (see Figure 5.7 (b)). The DOMINATED message of node 202 was overheard only by nodes 203 and 102. The dominator message from 201 was overheard by nodes 101, 102 and 202. Therefore, node 202 was elected as dominator by node 203 and 201 by 101. There are different reasons for this suboptimal CDS. First, a lot of control messages have been lost. Node 201 has a bad link

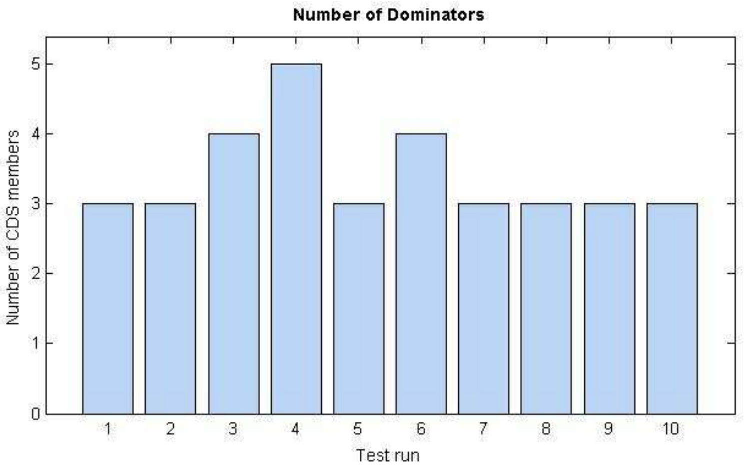


Figure 5.6: Number of dominators in each run.

to node 301. Therefore, only a few HELLO packets have arrived. Also other links performed badly. The main problem was that all neighbor nodes of the base station (301, 302, 303) did not overhear each other. These two reasons lead to neighbor tables on the nodes which do not reflect the real network topology. This leaded to an unexpected backbone.

We have also measured the median path length from all nodes, which are all at minimum n hops away from the base station. Considering the first column of the grid (nodes 101, 102, 103), in all of the ten cases the path length to the base station is 3 hops. In the second column (nodes 201, 202, 203) the backbone length is between 2 and 3 hops (in average is 2.7 hops) and in the third column (301, 302, 303) we measured between 1 and 3 hops (in average 1.9 hops). The nodes in the first column were all reachable with the minimum number of required hops ($n = 3$). The backbone length of the second ($n = 2$) and the third ($n = 1$) column is not optimal.

Evaluation of the Link Repair Algorithm

In this experiment we have configured a network that supports the simulation of link breaks. Therefore, we again have built a tree-like network. In Figure 5.8 you can see a redundant link between nodes 0405 and 0403. Either node 0304 or 0504 is initially elected as DOMINATOR node in dependence of its energy level. After the path has been built, either node 0304 or 0504 has been manually switched of (see Figure 5.9) to produce a link break. We have repeated this experiment 10 times. Every ten seconds a data packet with four bytes payload has been sent to the base station. Before the link break has occurred, 6 DOMINATOR nodes have been elected in every run. After the link break has been repaired we got seven dominator nodes in each test. In all cases the LINK BREAK message was kept always in the left branch of the tree.

We have measured how long it takes to repair the path. In Figure 5.10 for each run a bar indicating two different times is shown. The shorter one is the time which is spent between the

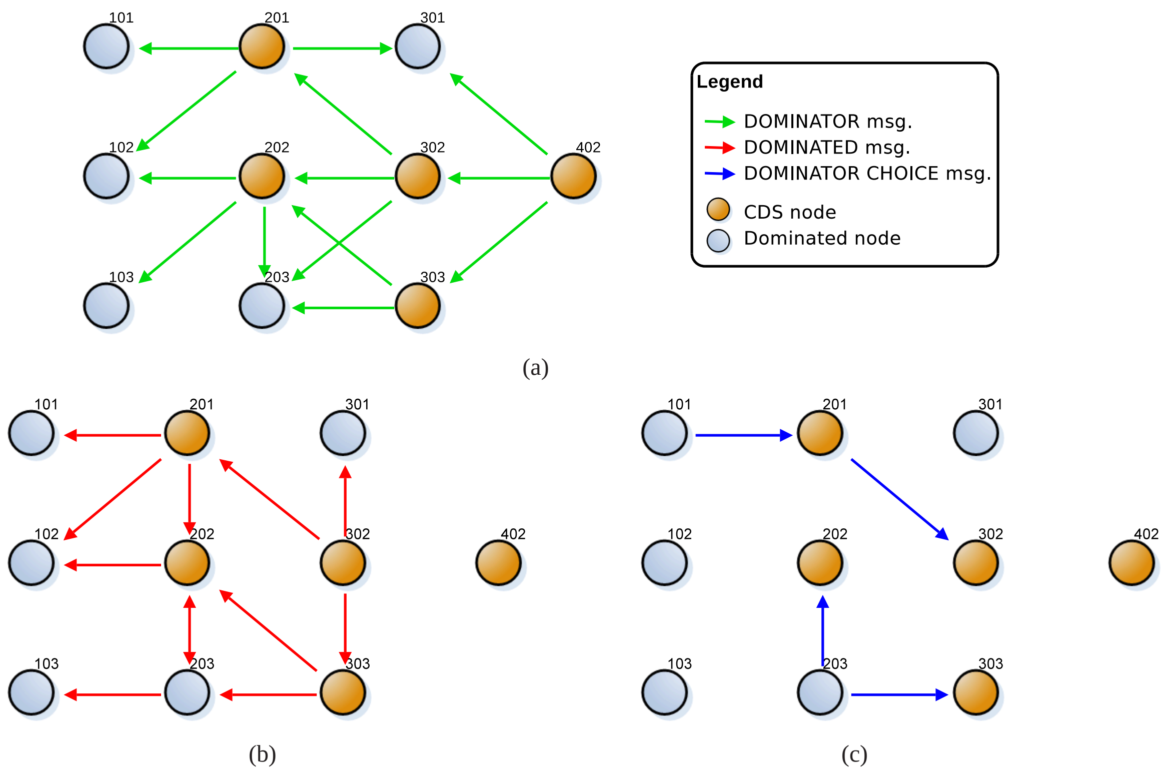


Figure 5.7: Analysis of the communication messages sent in the run producing a suboptimal backbone.

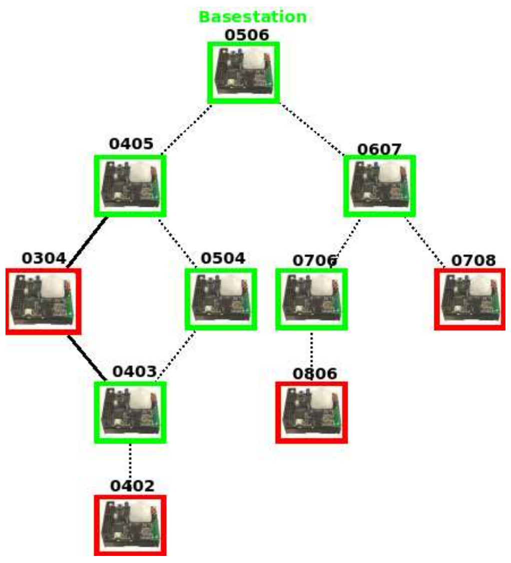


Figure 5.8: Test grid before link break.

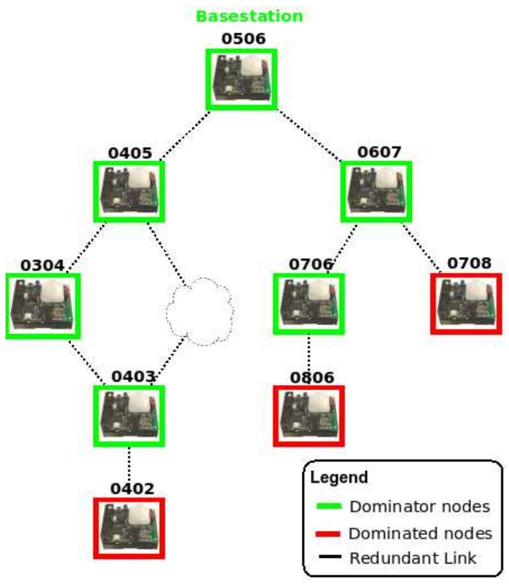


Figure 5.9: Test grid after link break.

moment node 0403 broadcasts the first link break message until the moment node 403 receives a link repair message. In average this takes about 0.25 s. The delay until a link break message is sent is between 0.1 s and 0.2 s. The delay until a link repair message is sent at maximum 0.1 s. Two link break messages are needed to reach node 405 which is the closest dominator node with a valid route to the base station. Two link repair messages are required to repair the link break at node 0403. In theory a repair time between 0.2 s and 0.6 s is expected if all nodes are awake during the link-break period. In some cases 2.7 s have been needed. This is explained as follows: If node 304 or node 504 is sleeping, no messages are forwarded. The link break message is retransmitted until one of these two nodes wakes up and forwards the message.

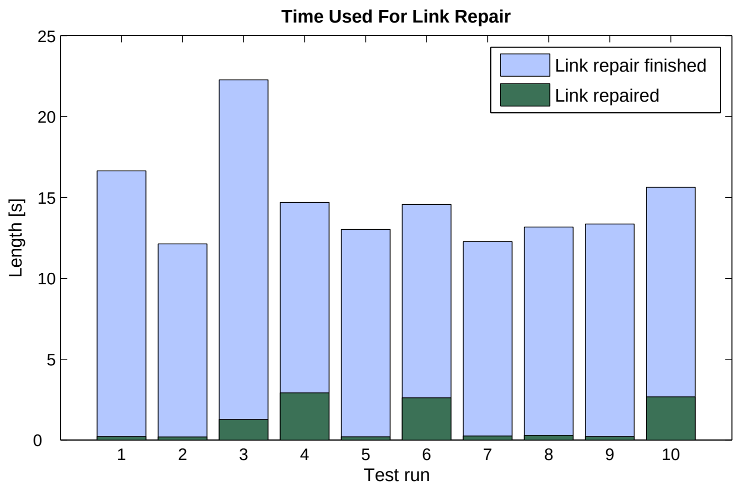


Figure 5.10: Time used to repair the link break.

The second time which is depicted in 5.10 is the time between the sending of the first link break message and the stop of sending any link repair or link break messages. The average total repair time has been 14.7 s. As our main goal was to prevent a growing amount of CDS nodes the link-break origin, in our case node 403, does not forward the link repair message. Node 402 stays in link break state, which means link break state, until node 403 wakes up again and sends a link break message to nodes 504 or 304. This message is then answered immediately by either node 504 or 304 and forwarded down to node 403. This node continuous sending link repair messages five times to ensure that each neighbor node has overheard this message. The sleep cycle is 10 s. Adding the four retransmission delays which lasts at maximum $4 * 1.05s = 4.2s$ to this $10s$ and the actual repair time at maximum $0.6s$ we get $14.8s$. Accordingly, the 14.7 s used until the last link repair or link break message has been sent is realistic.

5.2.2 Long Sleep Cycles Test

After testing the basic functionality we have run the network for a long period. We are interested in the percentage of time the nodes are sleeping over a longer term of operation. Because we need the logged information from the nodes, we have connected the nodes over a serial cable to

Node ID	Sleeping [%]	Times elected as dominator	Dominator by link break	link break detected
101	80	0	0	17
301	60	5	0	0
102	42	4	4	1
203	31	4	0	0
201	22	7	1	3
302	22	10	0	1
202	18	11	0	0
303	15	14	0	0
402	0	16	0	0

Table 5.2: Results of CDS long operation time evaluation.

a PC. The network is setup according to Figure 5.4. This time the runs last 8 hours. The timer configuration is set according to Table 4.2.

One long cycle lasts about 29 minutes (1769.4 s). A dominated node should, theoretically, be sleeping around 80% of this time. During the 8 hours of testing the network is reestablished 16 times. Table 5.2 shows how long each node has been sleeping during the whole experiment. The third column shows how often a node has been elected as dominator over all. The fourth column indicates how often a node has been elected as dominator after a link break has occurred and the fifth column is the counter how often a node has detected a link break. Node 101 which is never elected as dominator sleep for 80% of the whole test run. We see that the more often a node is elected as dominator, the lower is the percentage of time it is sleeping. The number of times nodes 102 and 203 have been elected as dominator is equivalent, but their sleeping time is different. The reason is that node 102 has been elected as dominator after a link break while 203 was dominator for the whole long-sleep period.

Node 101, which has a poor connectivity, detects 17 link breaks. Having analyzed the logging data we can see that lots of HELLO messages have not been overheard by node 101. Therefore, link break messages are sent. This leads to low sleeping times for its neighbors (nodes 102 or 201), as they are elected as dominator by the link repair mechanism. To prevent such a behaviour nodes with bad links could be ignored after a predefined number of link breaks. Nodes with erroneous link, for example because of a failure in hardware or software, would be ignored and could so not flood the network with link break messages. An other solution would be using acknowledged data packets. This would guarantee a relayable both-way link. So only nodes with acknowledged HELLO messages would appear in the neighbour table.

We have shown that the energy-based CDS algorithm performs well on a real-world hardware environment. CDS is able to reduce the energy consumption of network nodes. By reestablishing the network periodically CDS leads to a constant energy distribution over the whole network. The network repair algorithm makes the CDS stable and reliable in case of node failures.

5.3 DELTA Localization Evaluation

The localization performance has been evaluated in two steps. Initially, simulator test were done. Different localization algorithms have been implemented and tested with simulated data input. The most promising algorithm has then been implemented and tested in a real-world wireless sensor network using the ESB nodes.

5.3.1 Performance Evaluation in the Simulator

The three localization methods Linear Least Square, Conjugate Gradient Descend and Simplex Downhill, presented in Subsection 3.3, have been evaluated in MATLAB. For the evaluation, the sensor nodes have been arranged in a square with a side length of 125 cm. Four nodes have been placed at positions [125, 125], [125, 250], [250, 125] and [250, 250]. 200 events, randomly distributed within the square area, have been localized. As mentioned before Conjugate Gradient Descend and Simplex Downhill need initial values to start the computation. The starting simplex for the computation of the event position using Simplex Downhill is located at the center of area of the sensing nodes and their measurements. For Conjugate Gradient Descend the center of area of the sensing nodes is used as a starting point. Additional White Gaussian Noise (AWGN) was used to model noisy sensor measurements. The tests have been performed with six noise levels: 0%, 10%, 20%, 30%, 40% and 50%.

Figure 5.11 illustrates the results. The confidence intervals are not depicted as they are too small. If there is no noise, all methods perform equally well. For Linear Least Square the distance error as well as the signal strength error increase much faster than for Simplex Downhill and Conjugate Gradient Descend. The distance error produced by Linear Least Square stays constantly at about 40% of the transmission range. The error for the signal amplitude is even larger, it is more than 80% of the emitted signal strength. Simplex Downhill and Conjugate Gradient Descend in contrast show low error rates up to a noise of 20%. After that the distance error as well as the signal error start to increase. The signal amplitude error of Simplex Downhill is slightly lower than the error of Conjugate Gradient Descend for all noise levels.

The big difference between the performance of Linear Least Square and Simplex Downhill can be explained by analyzing Figure 5.12. The figures show the distance errors of Simplex Downhill and Linear Least Square with a noise level of 10% and 40%, respectively. Only 50 out of the 200 estimations are depicted for better readability. The distance error is highlighted by showing the line between the exact event position and the estimated event position. The figures on the left show estimation errors of Simplex Downhill with a noise of 10% and 40%, respectively. On the right the according errors for Linear Least Square are depicted.

Comparing the localization estimations of Simplex Downhill and Linear Least Square with a noise level of 10% we can see that the position estimations of Simplex Downhill are only little affected by the noise. on the other hand all estimations of Linear Least Square tend towards the center of the area observation. The figures at the bottom of 5.11 show a similar situation for Linear Least Square with a noise level of 40%. The distance error for Simplex Downhill

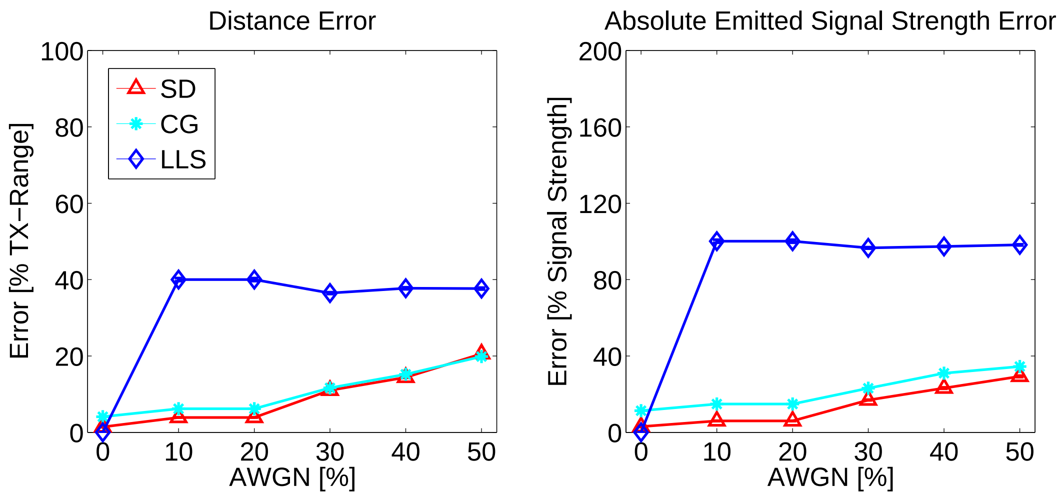


Figure 5.11: Evaluation of the localization accuracy of Simplex Downhill (SD), Conjugate Gradient Descend (CG) and Linear Least Square (LLS).

increases a little with a noise level of 40%.

The performance of Linear Least Square can be improved by over-determining the system as shown in Figure 5.13. Two additional nodes have been added at positions [175, 125] and [175, 250]. Instead of using the sensor readings of four nodes, six nodes are used for computing the position and emitted signal strength of the events. Using Linear Least Square, the distance error and the amplitude error start increasing at a noise level of 20% and reach an error of 40% considering distance error and a noise level of 50%. Like the distance error, the amplitude error is also increasing slower than in the system with four nodes. The error starts increasing with a noise level of 20% and reaches an error rate of 80% at a noise level of 50%. The performance of Simplex Downhill and Conjugate Gradient Descend is only little improved by the over-determined system.
Figure 5.14 visualizes the distance errors computed with the over-determined system. Again the position estimations of Linear Least Square tend to the center of the observation area. The estimation of the position with a noise of 10% is much better compared the results in Figure 5.14. Also the performance of Simplex Downhill is slightly improved by the two additional sensor nodes.

To conclude, the performance of Linear Least Square in an over-determined system is better, but does not reach the estimation quality of Simplex Downhill or Conjugate Gradient Descend. If additional sensor nodes are used for the computation this causes more traffic. Furthermore, denser networks are required to support Linear Least Square. Therefore, our choice has been restricted to Simplex Downhill or Conjugate Gradient Descend. Simplex Downhill is simple in implementation and usage. It requires little more time to terminate, but also need less storage. The performance is more or less the same as the one of Conjugate Gradient Descend. We have

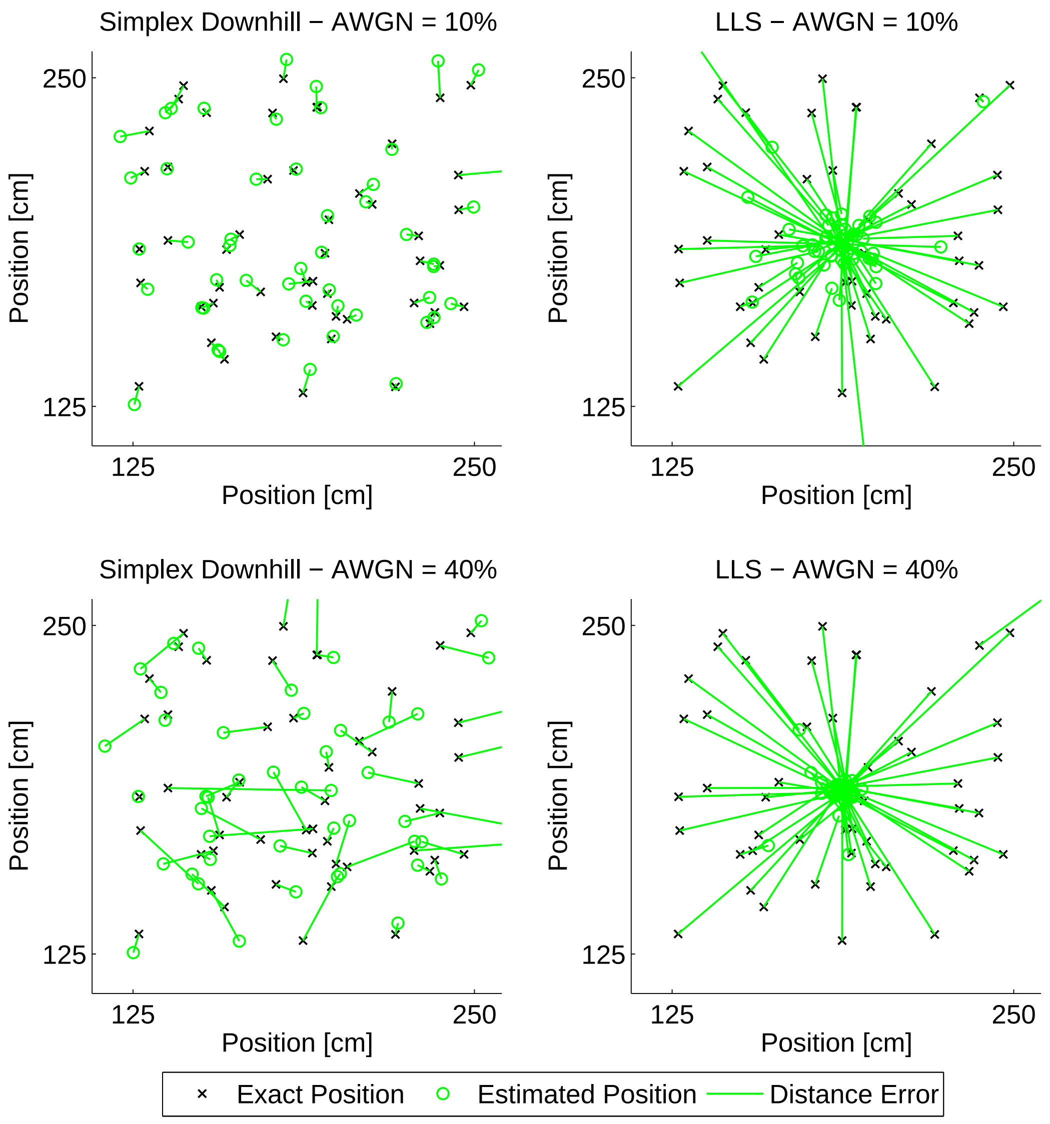


Figure 5.12: Distance error visualization for Simplex Downhill and Linear Least Square (LLS).

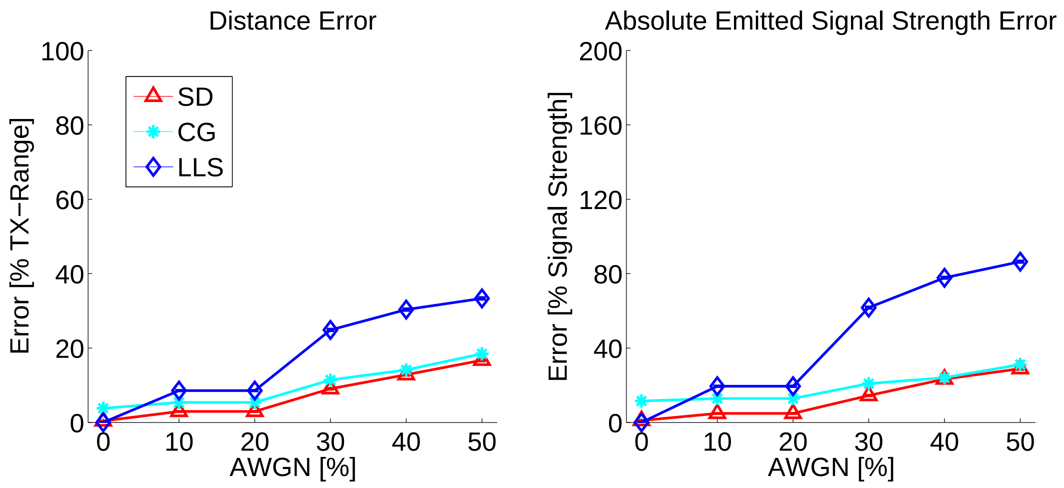


Figure 5.13: Evaluation of localization and parameter estimation accuracy using 6 sensor nodes for Conjugate Gradient Descent (CG), Simplex Downhill (SD) and Linear Least Square (LLS).

chosen Simplex Downhill for real-world tests and implementation on the ESB nodes.

5.3.2 Real-World Evaluation of Simplex Downhill

The localization of light events using Simplex Downhill has been evaluated in a real-world environment. In these tests four ESB nodes localize an event and compute the amplitude of the emitted light signal. The four nodes are set up in a square in a darkened room. As in the simulator tests, the distances between each pair of nodes is 125 cm. The positions of the nodes N_i are $N_1(125, 125)$, $N_2(250, 125)$, $N_3(250, 250$ and $N_4(125, 250)$. The orientation of the nodes has been calibrated such that the light sensor of the nodes points into the direction of the event. This is necessary because the light sensor is not optimally placed on the sensor boards. Each node is connected to a computer over a serial cable for logging purpose. The inputs for the computation as well as the localization results are logged for later recomputations and comparison. The light sources have been placed at 6 different positions, which are $P_1(188, 188)$, $P_2(188, 219)$, $P_3(188, 250)$, $P_4(219, 219)$, $P_5(219, 250)$ and $P_6(250, 244)$. The experiment setting is shown in Figure 5.15.

For the evaluation of Simplex Downhill five different bulbs have been used. We have chosen bulbs with uniform light emission in all directions. We use bulbs with an output of 25 Watt, 40 Watt, 60 Watt, 75 Watt and 100 Watt. The bulbs are placed into a holder standing on the floor.

Each bulb is tested at each event position. For each combination of bulb and position 50 Simplex Downhill computations are performed. The tests have been done twice. In the first run the following parameters for the Simplex Downhill computations have been used:

- **DIM**: 3.
- **NMAX**: 160

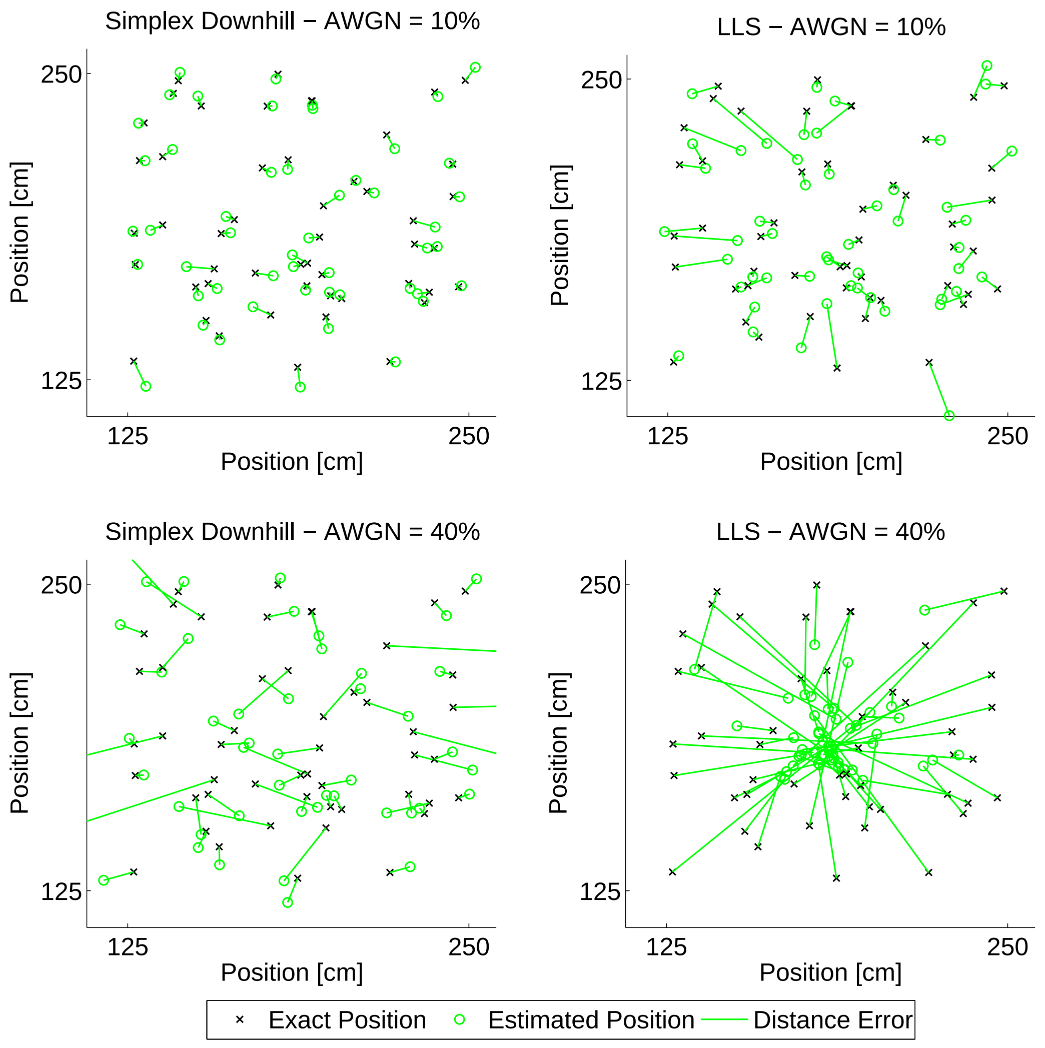


Figure 5.14: Distance error visualization of Linear Least Square and Simplex Downhill in an over-determined system.

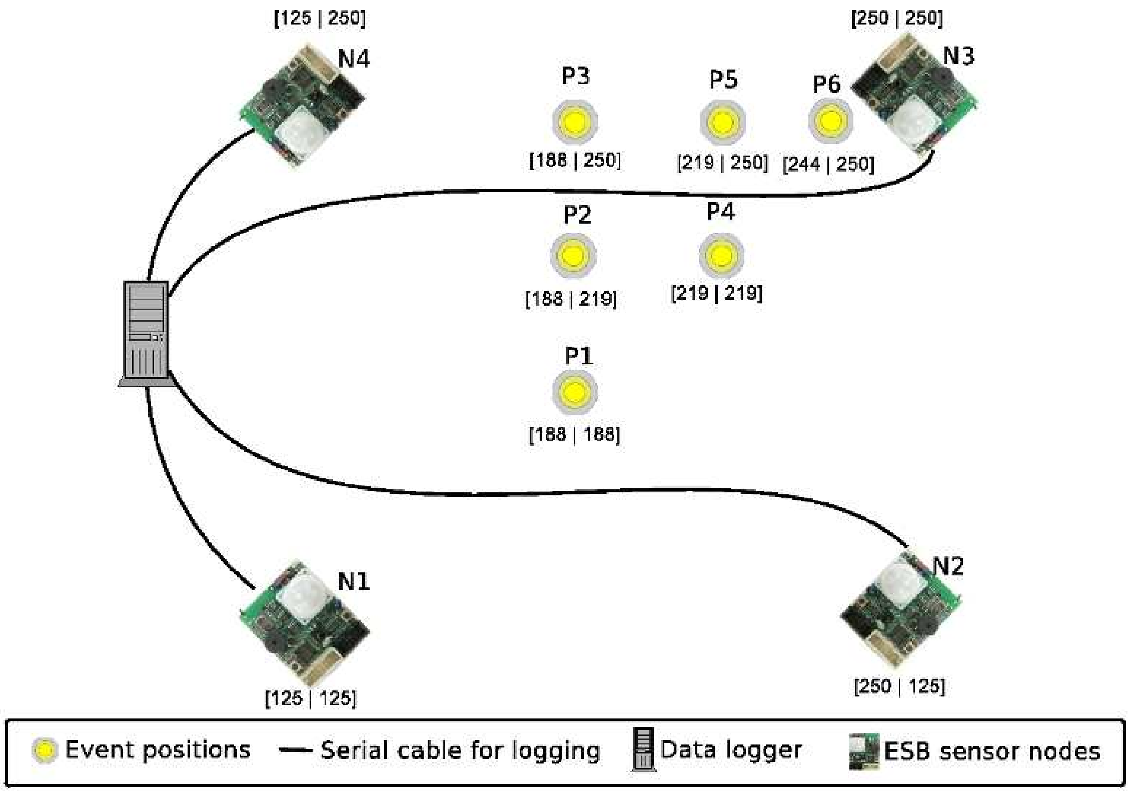


Figure 5.15: Experiment settings for the real-world localization tests.

	25 Watt		40 Watt		60 Watt		75 Watt		100 Watt		Over all	
Position	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
P_1	6.3	0.9	4.1	0.1	0.6	0.1	2.1	1.0	1.4	0.1	2.9	2.1
P_2	6.2	0.5	8.8	0.4	10.6	0.2	9.4	0.1	8.4	3.9	8.7	2.3
P_3	3.9	0.6	14.9	3.2	16.6	5.2	5.3	6.4	8.8	6.6	9.7	7.1
P_4	3.7	1.6	5.0	1.7	3.5	1.4	5.7	1.8	4.0	2.0	4.4	1.9
P_5	17.8	3.6	15.1	4.6	12.1	6.6	18.4	4.3	13.1	7.1	15.4	6.0
P_6	18.4	0.1	20.9	0.2	17.4	0.5	23.1	15.0	32.9	4.8	22.5	8.8

Table 5.3: Average localization error μ and standard deviation σ for each event position.

- **FTOL**: $1.0E{-}2$
- **NO_OF_REF_POINTS**: 4
- **COMPUTE_WIHT_AMPLITUDE**: Yes, amplitude is computed in Simplex Downhill.

The meanings of the different parameters have been presented in Section 4.3.

In the second run we have used exactly the same settings and the same testbed. Instead of computing the event intensity amplitude in Simplex Downhill we have computed it in a second step to reduce the problem dimension. After the computation of the event location the event intensity can be computed by the given distribution model. In the following we call the results from the tests with the amplitudes computed in Simplex Downhill *first run* results and the ones without the amplitude computed in the Simplex Downhill *second run* results. The results of both runs are comparable. For the evaluation of the position estimation and the amplitude estimation the results from the first run are used. In Subsection 5.3.2 we compare the performance of Simplex Downhill of both runs.

Position Estimation Accuracy

Table 5.3 shows the localization errors for each bulb and event position, where μ represents the mean distance error and σ the standard deviations over all 50 position estimations from the first run results. The last column shows the average localization error over all bulbs. P_1 to P_6 are the event positions shown in Figure 5.15. The error is defined as the Euclidean distance between the event location P_i and the estimated positions E_k computed by Simplex Downhill.

Figure 5.16 visualizes all distance errors. Each color represents an event position, which is represented by a circle. The event positions estimated by Simplex Downhill are represented by **x** symbols. The error is represented as the lines between the **x** and **x**.

Considering Table 5.3 and Figure 5.16 it can be seen that the closer the events are positioned to the border of the sensor grid, the higher is the mean distance error. The maximal mean error for event positions P_1 to P_5 is 18.4 cm, sensed with the 75 Watt bulb. At event position P_6 the distance errors are higher. The performance of Simplex Downhill decreases if the event position is getting closer to a node. In these areas the range of correct solutions is small, whereas the risk

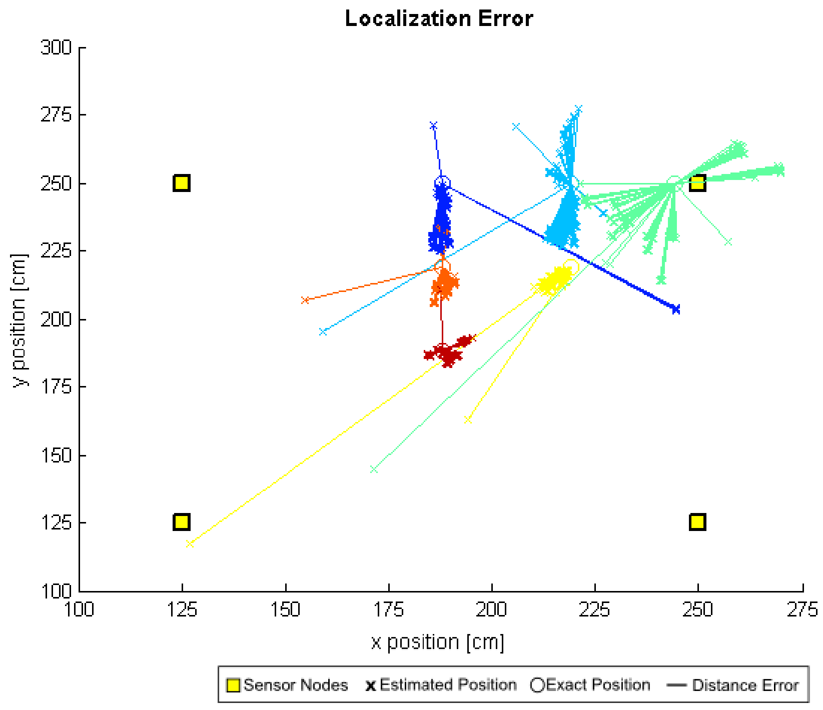


Figure 5.16: Visualization of the event position estimation error.

of finding local minima is high. However, due to the implementation of DELTA sensor nodes will arrange themselves around the event location. Accordingly, events are rather located near the center of the observing sensor group. Table 5.3 shows that the intensity of the light has only little influence on the position errors of the position estimations.

Excluding event position P_6, the position estimation of Simplex Downhill is quite robust and works well for different bulbs. Considering the distance of 125 cm between the sensor nodes, a maximal mean distance error of 18.4 cm with a variance of 4.3 cm is acceptable (result without event position P6).

Amplitude Estimation Accuracy

For the classification of the detected events some characteristics about the event are needed. Therefore, Simplex Downhill not only estimates event positions but also their amplitudes. First we present the evaluation data and discuss them in the next step: Table 5.4 shows the mean (μ) and the standard deviations (σ) of the estimated emitted light amplitude for each event position and bulb based on the data computed by Simplex Downhill for the first run results. The last column shows the mean of estimated amplitudes and their standard deviations over all event positions. The mean amplitude of events with a 25 Watt bulb is $17.7e^5$ Hz. The mean intensity of the 40 Watt bulb is with an estimated intensity of $28.8e^5$ Hz around $10.0e^5$ Hz higher. For the 60 Watt bulb an amplitude of $47.4e^5$ Hz was estimated, for 75 Watt bulb $61.6e^5$ Hz and for

	P_1		P_2		P_3		P_4	
Bulb [Watt]	μ	σ	μ	σ	μ	σ	μ	σ
25	$16.9e^5$	$0.88e^5$	$16.7e^5$	$0.23e^5$	$13.2e^5$	$0.05e^5$	$17.8e^5$	$1.30e^5$
40	$30.0e^5$	$0.06e^5$	$29.1e^5$	$0.27e^5$	$27.5e^5$	$1.03e^5$	$31.1e^5$	$2.32e^5$
60	$47.8e^5$	$0.14e^5$	$46.7e^5$	$0.20e^5$	$42.5e^5$	$2.73e^5$	$49.5e^5$	$3.06e^5$
75	$59.3e^5$	$0.19e^5$	$59.0e^5$	$0.26e^5$	$51.5e^5$	$6.10e^5$	$63.2e^5$	$5.39e^5$
100	$82.5e^5$	$0.19e^5$	$80.4e^5$	$11.6e^5$	$71.6e^5$	$7.63e^5$	$86.0e^5$	$8.58e^5$

	P_5		P_6		All positions	
Bulb [Watt]	μ	σ	μ	σ	μ	σ
25	$18.7e^5$	$2.62e^5$	$19.8e^5$	$0.04e^5$	$17.1e^5$	$2.47e^5$
40	$27.0e^5$	$10.5e^5$	$28.4e^5$	$0.07e^5$	$28.8e^5$	$4.53e^5$
60	$49.8e^5$	$11.1e^5$	$48.1e^5$	$0.54e^5$	$47.4e^5$	$5.35e^5$
75	$73.8e^5$	$10.3e^5$	$65.4e^5$	$16.3e^5$	$61.6e^5$	$10.8e^5$
100	$86.0e^5$	$15.1e^5$	$124.0e^5$	$24.0e^5$	$88.5e^5$	$21.4e^5$

Table 5.4: Computed amplitudes in Hz for each event position.

the 100 Watt bulb $88.5e^5$ Hz. The standard deviations increase with the emitted light intensity of the bulbs.

Figure 5.17 visualizes the distribution of the amplitude estimations for each bulb as a box-plot. The line in the middle of the boxes is the sample median. The tops and the bottom of the box are the 75th and 25th percentiles. The length of the box describes the interquartile range. The whiskers are the lines looking like a ”T” extending the boxes. They describe the values which are within 1.5 times the interquartile range beginning at the end of the box. The values displayed with a red ”+” are the outliers. We see that the interquartile ranges are small. The number of outliers is higher for the events with 75 Watt and 100 Watt bulbs. This indicates that the events with 75 Watt bulb and these with the 100 Watt bulb are less disjoint.

For a classification of events it is important that the spectrums of the amplitudes of different events (i.e. different bulbs) are disjoint. This condition is fulfilled in most of the situations shown in Table 5.4. Only at position P_5 the computed light amplitudes are not fully disjoint: The estimated amplitudes between the events with the 25 Watt bulb and the 40 Watt bulb and the events with the 40 Watt and the 60 Watt bulb overlap. Furthermore, classifications of the 75 Watt and 100 Watt bulbs might cause some systematic false classifications comparing the spectrums of μ and σ in the last column in Table 5.4. Figure 5.17 illustrates this too. Some outliers of the 75 Watt bulb are in the range of the box of the 100 Watt bulb. Over all we can say that correct classification should be possible in most cases.

We have seen that the amplitude and also distance error increases the closer the event comes to a sensor node (see Figure 5.16). We have used a ranking table to visualize this. In theory the amplitude estimation should be the same at each event position. In Table 5.5 we show that

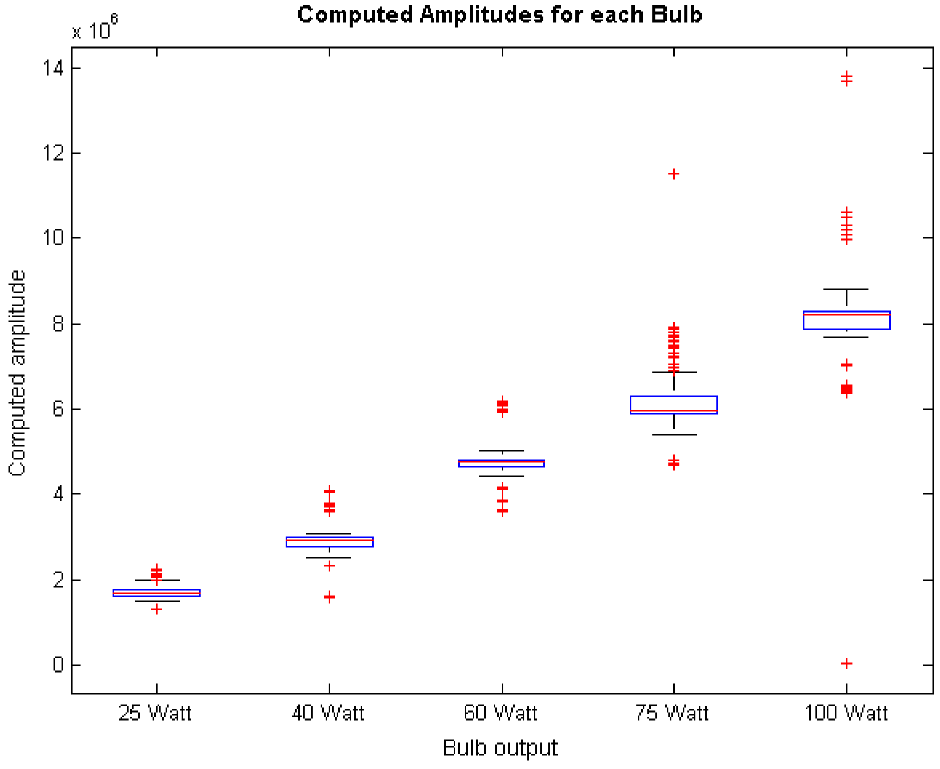


Figure 5.17: Boxplot of the estimated emitted signal strength amplitudes using Simplex Downhill.

	25 Watt	40 Watt	60 Watt	75 Watt	100 Watt	Total rank
P_3	6	5	6	6	6	29
P_2	5	3	5	5	5	23
P_1	4	2	4	4	4	18
P_5	2	6	1	1	2	12
P_6	1	4	3	2	1	11
P_4	3	1	2	3	2	11

Table 5.5: Ranking of the amplitudes for each position.

the amplitude estimation error also depends on the event position. For each bulb the amplitudes at the different positions have been analyzed. We ranked in Table 5.5 each bulb. The position where the highest amplitude for a bulb was estimated gets the highest rank 1 the position where the lowest amplitude was estimated gets the lowest rank 6. The last column is the total score. It is the sum of ranks for each bulb at each position. The position with the lowest estimated amplitudes for each bulb gets the highest value, the one with the highest estimated amplitudes gets the lowest value.

Table 5.5 shows that the error increases for the amplitude estimations the closer the event is to a sensor node. Events at the positions P_4 and P_6 have the lowest ranking, event P_3 the highest. This means that at the positions P_4 and P_6 the amplitudes have been estimated higher for the same bulb as at the other positions and at position P_3 the amplitudes have been estimated lower for the same bulb at the other positions. For the events at the positions P_6, P_4 and P_5 which are closer to node N_3, higher amplitudes for the same bulb have been sensed compared the events at the positions P_3, P_2 and P_1. As expected not only the localization estimation error increases the closer the event is to a sensor node but also the amplitude estimation error.

Evaluation of Simplex Downhill without Amplitude Computation

During the evaluation of the localization accuracy with Simplex Downhill, no result was computed in some cases because of too many iterations of Simplex Downhill. As mentioned before the number of iterations is limited to prevent a system reset by the watch dog. Also we have limited the number of iterations to account for some delays which could lead to problems in communications of the group organization algorithm of DELTA. To reduce the complexity of Simplex Downhill and make it faster, we have reduced the problem dimension. We have removed the amplitude estimation from Simplex Downhill and have computed only the event position. The amplitude estimation has been done afterwards, as described in Section 4.3. This gives us the freedom to compute the amplitude only if it is used for further action. We run the same tests again with the modified Simplex Downhill as before in the real-world environment.

The estimations of event position computed with the modified Simplex Downhill are comparable with the results from the first run. An interesting difference is the number of iterations. Table 5.6 lists the number of iterations which have been used in the first run, where the ampli-

	Run 1		Run 2	
Bulb [Watt]	μ	σ	μ	σ
25	53	24	31	7
40	62	18	32	9
60	60	21	35	11
75	57	27	32	8
100	52	21	31	7

Table 5.6: Comparing Simplex Downhill computing amplitude (Run 1) and Simplex Downhill not computing amplitude (Run 2).

tude has been computed in Simplex Downhill and in the second run, where the amplitude has not been computed in Simplex Downhill. For both runs the number of iterations per bulb in the average (μ) and the mean (σ) is shown. In the first run we got an over all average of 56 iterations, where as in average 32 iterations are sufficient to for compute the events position alone.

This shows that the performance of Simplex Downhill can be improved by removing the amplitude estimation.

5.3.3 Parameter Optimization

The two test series presented in 5.3.2 have been performed with the same parameter configurations. To find out if there is a more optimal parameter configuration for the localization task we let running Simplex Downhill with different parameter settings. As it is a time consuming task testing localization in a real-world environment we have ported the simplex code to a PC environment and used the real-world data from the first test serie as input for further optimization steps. In the first run we have not only logged the results of the Simplex Downhill computation but also the sensor readings. This raw data has been used as input for Simplex Downhill. Our main focus is on finding the optimal parameter configuration, especially considering the settings of the *FTOL* termination condition. Therefore we have tested Simplex Downhill and the modified Simplex Downhill without amplitude computation with *FTOL* $1.0e^{-2}$, $1.0e^{-4}$, $1.0e^{-6}$, $1.0e^{-8}$ and $1.0e^{-10}$. The other parameters have not been changed. To find an optimal configuration parameter we define the following criteria:

1. The mean of the amplitude spectrums of each bulb have to be disjoint.

2. The mean distance error has to be minimized.

3. The number of aborted computations because of too many iterations should be minimized ($NMAX = 30$).

Table 5.7 shows the results of the evaluation of Simplex Downhill with different parameters based on the input data from the first real-world run. The first row shows which Simplex Downhill version has been used. As mentioned above, the amplitude can be computed in Simplex

Amplitude computation	FTOL	Aborted Computations	μ	σ	Amplitudes disjoint
in SD	$1.0e^{-2}$	53	11.46	9.04	No
in SD	$1.0e^{-4}$	149	11.18	9.37	Yes
in SD	$1.0e^{-6}$	273	9.41	8.66	Yes
in SD	$1.0e^{-8}$	632	8.92	7.85	Yes
in SD	$1.0e^{-10}$	632	8.92	7.85	Yes
offline	$1.0e^{-2}$	35	10.94	8.03	Yes
offline	$1.0e^{-4}$	97	10.4	7.75	Yes
offline	$1.0e^{-6}$	97	10.4	7.75	Yes
offline	$1.0e^{-8}$	335	10.37	7.95	Yes
offline	$1.0e^{-10}$	335	10.37	7.95	Yes

Table 5.7: Results of running Simplex Downhill (SD) with different parameters.

Downhill itself or afterwards. The column μ shows the mean position estimation error over all computations and σ is the standard deviation. The last column shows if the means of amplitude estimations between every pair of bulbs are disjoint.

Apart from the first row all amplitude computations are disjoint. The amplitude estimations between the 60 Watt and the 75 Watt bulb and between the 75 Watt and the 100 Watt bulb overlap. This means that the mean of the amplitude estimation of a bulb type A plus the standard deviation is higher than the mean of the amplitude estimation of a bulb type B minus the standard deviation. The optimal results we got by using the modified Simplex Downhill with $FTOL = 1.0e^{-4}$. There is a trade off between the estimation accuracy and the number of aborted computations. Simplex Downhill returns accurate results with $FTOL = 1.0e^{-4}$ for the offline amplitude computation as well as for the computation of the amplitude of Simplex Downhill.

We see that the optimized parameters lead to a better performance of Simplex Downhill compared with the settings used in the real-world test runs. The means of the estimated amplitudes for each bulb are disjoint and the localization errors are smaller with the optimized parameters. The number of aborted computations is slightly higher. From the raw sensor readings from the two existing real-world tests (see Section 5.3.2) we have derived the two sets of estimation results with the following settings:

- **DIM**: 2.
- **NMAX**: 160
- **FTOL**: $1.0E{-4}$
- **NO_OF_REF_POINTS**: 4
- **COMPUTE_WIHT_AMPLITUDE**: No, amplitude is computed afterwards.

These two result sets will be used in the next section for evaluation of the classification algorithms. Figure 5.18 gives an overview of the test sets and how they interpend.

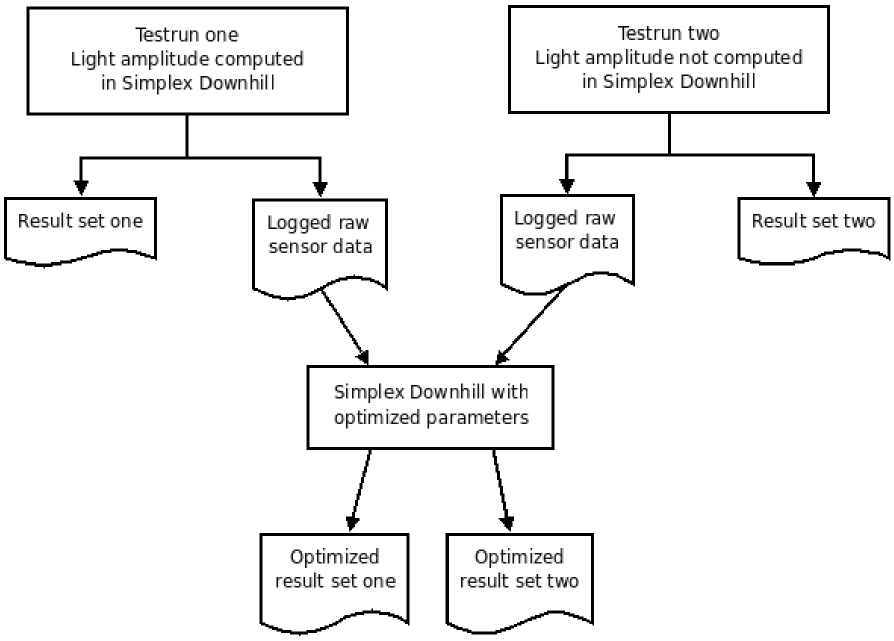


Figure 5.18: Result sets from real-world Simplex Downhill evaluation.

5.4 Delta Classification Evaluation

5.4.1 Simulator Tests

In a first step we have evaluated the proposed classification algorithms minimal distance classifier and Bayesian Classifier in the MATLAB environment. This allows us to evaluate the two classification algorithms without side effects. The classifier with the better performance is implemented on the real-world hardware in a second step. The results from the simulator tests are also used as reference to compare performance of the real-world implementation.

As input for the event classification we have used the one-dimensional real-world input data computed with the optimized Simplex Downhill configuration parameters described in Subsection 5.3.3. As mentioned above we have two sets of real-world data. Figure 5.19 describes how the classification is applied. We have used the optimized data from the first test serie as training set and the data from the second optimized test serie as test set. We have applied K-Means on this first set to compute the cluster characteristics. The results of K-Means are than used to find the parameters for the minimal distance classifier and the Bayes classifier. The two trained classifiers are then applied on the second optimized data test set. First we give a short overview over the characteristics of the two input sets. Table 5.8 shows the characteristics of the estimated amplitudes of test and training sets. μ_i represents the mean amplitude and σ_i the standard

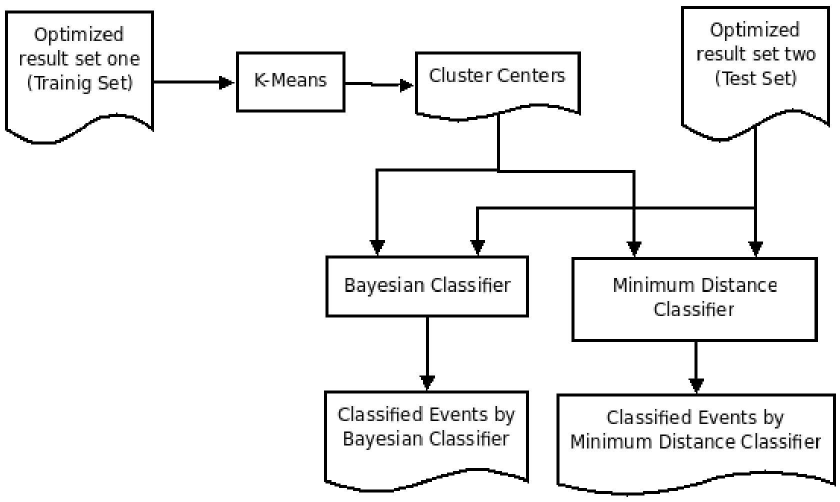

Optimized result set one (Trainig Set)
K-Means
Cluster Centers
Optimized result set two (Test Set)
Bayesian Classifier
Minimum Distance Classifier
Classified Events by Bayesian Classifier
Classified Events by Minimum Distance Classifier

Figure 5.19: Application of the classifier algorithms on the input data.

	μ_{25}	σ_{25}	μ_{40}	σ_{40}	μ_{60}	σ_{60}
Training set	$1.84e^6$	$0.60e^6$	$2.94e^6$	$0.27e^6$	$4.65e^6$	$0.32e^6$
Test set	$1.59e^6$	$0.15e^6$	$2.98e^6$	$0.55e^6$	$4.45e^6$	$0.27e^6$

	μ_{75}	σ_{75}	μ_{100}	σ_{100}
Train set	$6.00e^6$	$0.76e^6$	$8.44e^6$	$1.30e^6$
Test set	$5.64e^6$	$0.57e^6$	$8.09e^6$	$1.58e^6$

Table 5.8: Characteristics of train and test data set.

deviations of the amplitude where as i is the kind of the bulb in Watt. As already mentioned the averages of the amplitude estimation of each bulb are disjoint. The characteristics of both real-world tests are not exactly the same. The 25 Watt bulb has for example a mean amplitude estimation of $1.84e^6$ with a standard deviation of $0.60e^6$ in the training set where as in the test set the mean amplitude estimation is $1.59e^6$ and a standard deviation of $0.15e^6$. The diversity of the data sets is desired because we want to prove that the classification algorithm is flexible enough to be applicable on real-world data.

Table 5.9 shows the results of the classification using minimal distance classifier and Bayes classifier. Bayes classifier classifies with a success ration of 93.6% and minimal distance classifier wit 90.2%.

Some of the amplitudes computed by Simplex Downhill show high deviations. Regarding this fact the classification accuracy of Bayes classifier is quite high. We decided to implement Bayes classifier on the real-world hardware. It has not only a higher accuracy than the minimal distance classifier but is also usable with multi-dimensional data sets. minimal distance classifier is only

	Right	False	Success Ratio
MDC	1040	113	90.2%
BC	1079	74	93.6%

Table 5.9: Classification of one dimensional input with Bayes classifier (BC) and minimal distance classifier (MDC).

usable in circular clusters. If the elements of a cluster are in a non circular area they are probably classified wrong. Figure 5.20 shows two clusters in oval shapes. Figure 5.21 shows how the elements would be classified using minimal distance classifier. The two circles symbolize the two clusters. On the other hand, Bayes classifier is basically able to model such shapes. Thus, we have decided to implement Bayes classifier on the real-world hardware.

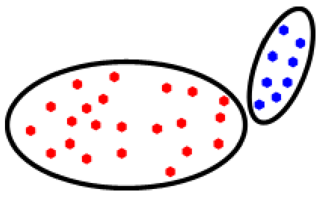


Figure 5.20: Correct classification.

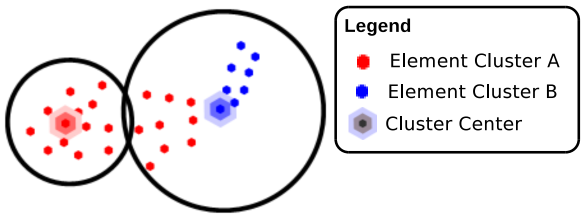


Figure 5.21: Classification with minimal distance classifier.

5.4.2 Real-World Tests

On the ESB nodes we evaluate the implementation of the one dimensional Bayes classifier using the same data sets as used in the one dimensional simulator tests. Therefore, we have written a script that sends the amplitudes of the training set to the ESB nodes over a serial cable where they are evaluated. The results computed on the ESB nodes are slightly different from the results obtained from MATLAB. Bayes classifier on ESB nodes has classified 89.5% correctly where as the MATLAB implementation does this in 93.6% of the cases. This can be explained by the differences in the implementation. On the ESB nodes only the type *float* with single precision is supported, whereas MATLAB computes results with higher precision. The rounding errors leads to different results. Nevertheless, we could show that Bayes classifier performs well on real-world hardware with limited resources.

Chapter 6

Conclusion and Outlook

In this chapter we give some final conclusion about the presented work. After this we present fields for further investigations considering the DELTA framework.

6.1 Conclusion

In this thesis we have presented extensions to the DELTA framework. The thesis covers four subject areas: Network organization, event detection, event localization and event classification. For each of these parts we have presented real-world implementations and evaluation on WSN nodes with limited power constraints. In the following section we present a conclusion for each evaluated area.

Network Organization: The Receiver-Based Backbone Construction algorithm supports DELTA with energy-efficient routing. We have shown that the algorithm is applicable to hardware with limited resources. Real-world tests have shown, that the basic functionality of the algorithm is working properly. In the network set-up phase the CDS is built as expected. In a second real-world experiment we have shown that the network is able to repair itself when a backbone node fails. We have also shown that, using the concept of backbone nodes, the network nodes can save energy of up to 80% of their lifetime by sleeping.

Event Localization: In this thesis Linear Least Square, Conjugate Gradient Descend and Simplex Downhill were investigated to be used for event localization. The localization estimations of Linear Least Square tends to the center of the sensing area. The quality of the localization estimations for Linear Least Square has been improved with an over determined system. Therefore, Linear Least Square needs sensor readings of more nodes than Conjugate Gradient Descend and Simplex Downhill to produce localization estimations similar quality. Simplex Downhill and Conjugate Gradient Descend have shown both similar performance. The Simplex Downhill algorithm has been implemented on the ESB nodes for further real-world evaluation. In the real-world evaluation of the event localization and of emitted signal strength estimation, Simplex Downhill produced reliable localization information. The closer the events are to a sensing node, the less precise is the localization estimation. The maximal localization error was 14.7% of the distance between neighboring nodes, neglecting the estimation of the event closest to the sensor node.

Event Classification: In this thesis we have presented a distributed self-learning event classification procedure for DELTA. The Simplex Downhill algorithm can not only be used for event position estimations but also for emitted signal strength amplitude estimations. The real-world tests with five different bulbs have shown that the computed amplitudes of each bulb are disjoint. Using this one-dimensional real-world data we have evaluated Bayes classifier and minimal distance classifier. The evaluation has shown that 93.6% of the events are classified correctly using a Bayes classifier and 90.2% of the events are classified correctly using minimal distance classifier. The implementation of Bayes classifier on the real-world wireless sensor nodes has shown a slightly lower accuracy. We have also presented a solution that allows adapting and updating the distributed event classification algorithm at runtime to classify new events.

Over all we have shown that it is possible to implement an energy-efficient event localization and classification framework on real-world wireless sensor node hardware.

6.2 Outlook

In this section we present areas for further investigations. We concentrate us on the evaluation of the framework. Evaluating real-world networks is a time consuming tasks. In this thesis we evaluated the basic functionality of the framework. There was no time left for further, more realistic, evaluation scenarios. In the following paragraphs we will present areas for future investigations in the three areas of network organization, event localization and event classification.

Network Organization: The receiver-based backbone construction algorithm was evaluated in small networks with up to 10 network nodes. Further real-world tests in larger networks could be done. In our evaluation we have used grid topologies. Tests with randomly distributed network nodes over larger areas could be done to get more information of the performance characteristics of this algorithm in a more realistic environment.

Event Localization: The real-world experiments of the localization were done in a network of four nodes. Events close to a sensor node produced higher localization errors. To prove that this error could be minimized in networks with more nodes the tests presented in this thesis could be done in larger WSNs. Further event localization in non-grid-like network topologies could be evaluated. A more complex real-world scenario such as a network with randomly distributed network nodes could be evaluated.

Event Classification: The implementation and evaluation of the event classification algorithm was done with light sensor readings. The evaluated classification algorithms would be able to handle multidimensional inputs. This allows classifying more complex events, as for example an event which emits light and sound. Therefore, the implementation has to be modified to support multidimensional classification. Furthermore, appropriate hardware equipped with more sensors could be used.

The Receiver-Based Backbone and the DELTA framework have not been tested running concurrently. To get a better overview over the performance of the presented extended version of DELTA a long time evaluation with larger networks and classification of more complex events would be necessarily. The backbone support has to be integrated into DELTA for these long time tests to see how event reporting over the CDS backbone and event detection, localization and classification perform together.

Bibliography

[1] T. Abdelzaher, B. Blum, Q. Cao, Y. Chen, D. Evans, J. George, S. George, L. Gu, T. He, S. Krishnamurthy, L. Luo, S. Son, J. Stankovic, R. Stoleru, and A. Wood, "Envirotrack: Towards an environmental computing paradigm for distributed sensor networks," in *Proc. of the International Conference on Distributed Computing Systems*. Los Alamitos, CA, USA: IEEE Computer Society, 2004, pp. 582–589.

[2] I. Cidon and O. Mokryn, "Propagation and leader election in multihop broadcast environment," in *Proc. of the 12th International Symposium on Distributed Computing*, London, UK, 1998, pp. 104–119.

[3] B. Clark, C. Colbourn, and D. Johnson, "Unit disk graphs," *Discrete Math.*, vol. 86, no. 1-3, pp. 165–177, 1990.

[4] *User Guide for Scatterweb*, Freie Universität Berlin, 2007. [Online]. Available: http://cst.mi.fu-berlin.de/projects/ScatterWeb/documentation/User_Guide.pdf

[5] "2-wire digital thermometer and real time clock," Freie Universität Berlin, Institute of Computer System and Telematics, 2007. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb-net/datasheets/DS1629.pdf

[6] "Electret condenser microphone unit," Freie Universität Berlin, Institute of Computer System and Telematics, 2007. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb-net/datasheets/ECM-10.pdf

[7] "Fresnel lens," Freie Universität Berlin, Institute of Computer System and Telematics, 2007. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb-net/datasheets/FLINSE.pdf

[8] "Polymetric infrared detector," Freie Universität Berlin, Institute of Computer System and Telematics, 2007. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb-net/datasheets/PIR.pdf

[9] "Product data sheet - movement / vibration switch cm 1800-1," Freie Universität Berlin, Institute of Computer System and Telematics, 2007. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb-net/datasheets/CM1800-1.pdf

[10] "Tsl245 - infrared light-to-frequency converter," Freie Universität Berlin, Institute of Computer System and Telematics, 2007. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb-net/datasheets/TSL245.pdf

[11] M. Friedman and A. Kandel, *Introduction to Pattern Recognition Statistical, Structural, Neural and Fuzzy Logic Approaches*. World Scientific Publishing, 1999.

[12] A. Galstyan, B. Krishnamachari, K. Lerman, and S. Pattem, "Distributed online localization in sensor networks using a moving target," in *Proc. of the third international symposium on Information processing in sensor networks*, New York, NY, USA, 2004, pp. 61–70.

[13] S. Guha, R. Murty, and E. G. Sirer, "Sextant: a unified node and event localization framework using non-convex constraints," in *Proc. of the 6th ACM international symposium on Mobile ad hoc networking and computing*, New York, NY, USA, 2005, pp. 205–216.

[14] S. Gyula, M. Maróti, A. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton, "Sensor network-based countersniper system," in *Proc. of the 2nd international conference on Embedded networked sensor systems*, New York, NY, USA, 2004, pp. 1–12.

[15] M. Heisenbuettel, T. Braun, M. Wälchli, and T. Bernoulli, "Optimized stateless broadcasting in wireless multi-hop networks." in *Proc. of 25th IEEE International Conference on Computer Communications*, Barcelona, Spain, 2006, pp. 1–12.

[16] C. Hu, S. Xu, , and X. Yang, "A review on interval computation - software and applications," *International Journal of Computational and Numerical Analysis and Applications*, vol. 1, pp. 149–162, 2002.

[17] T. Ishioka, "Extended k-means with an efficient estimation of the number of clusters," in *Proc. of the Second International Conference on Intelligent Data Engineering and Automated Learning, Data Mining, Financial Engineering, and Intelligent Agents*, London, UK, 2000, pp. 17–22.

[18] L. Kinsler, A. Frey, A. C. Sanders, and J. Sanders, *Fundamentals of Acoustic*. John Wiley and Sons, 2000.

[19] D. Li and Y. Hu, "Energy-based collaborative source localization using acoustic microsensor array," *EURASIP J. Appl. Signal Process.*, vol. 2003, no. 1, pp. 321–337, 2003.

[20] ——, "Least square solutions of energy based acoustic source localization problems," in *Proc. of the 2004 International Conference on Parallel Processing Workshops*, Washington, DC, USA, 2004, pp. 443–446.

[21] D. Li, K. Wong, Y. Hu, and A. Sayeed, "Detection, classification, and tracking of targets," 2002.

[22] J. Liu, M. Chu, J. Liu, J. Reich, and F. Zhao, “Distributed state representation for tracking problems in sensor networks,” in *Proc. of the third international symposium on Information processing in sensor networks*, New York, NY, USA, 2004, pp. 234–242.

[23] L. Luo, T. Abdelzaher, T. He, and J. Stankovic, “Envirosuite: An environmentally immersive programming framework for sensor networks,” *Trans. on Embedded Computing Systems*, vol. 5, no. 3, pp. 543–576, 2006.

[24] *Matlab Manual - Kmeans*, Mathworks, 2007. [Online]. Available: http://www.mathworks.com/access/helpdesk/help/toolbox/stats/index.html

[25] C. Maurice, H. Haussecker, and F. Zhao, “Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks,” *International Journal of High Performance Computing Applications*, vol. 16, no. 3, pp. 293–313, 2002.

[26] J. Nelder and R. Mead, “A simplex method for function minimization,” in *Computer Journal*, 1965, pp. 308–313.

[27] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical recipes in C. The art of scientific computing*. Cambridge: University Press, 1992.

[28] *Datasheet for TR1001 868.35 MHz hybrid transceiver*, RF Monolithics, 2008. [Online]. Available: http://www.rfm.com/products/data/tr1001.pdf

[29] X. Sheng and Y. Hu, “Energy based acoustic source localization,” Palo Alto, California, USA, 2003, pp. 285–300.

[30] ——, “Maximum likelihood multiple-source localization using acoustic energy measurements with wireless sensor networks,” *IEEE Trans. on Signal Processing*, vol. 53, no. 1, pp. 44–53, 2005.

[31] P. Skoczylas, M. Wälchli, and T. Braun, “Implementation of the delta object tracking algorithm on the esb sensor nodes,” 2006, -. [Online]. Available: http://www.iam.unibe.ch/publikationen/techreports/2006/iam-06-008/file/at_download

[32] J. Smith and J. Abel, “Closed-form least-squares source location estimation from range-difference measurements,” *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 35, no. 12, pp. 1661–1669, 1987.

[33] *MSP430x1xx Family Users Guide*, Texas Instruments, 2006. [Online]. Available: http://focus.ti.com/lit/ug/slau049f/slau049f.pdf

[34] S. Underwood, *mspgcc - A port of the GNU tools to the Texas Instruments MSP430 microcontrollers*, 2003. [Online]. Available: http://mspgcc.sourceforge.net/manual/

[35] M. Wälchli, T. Bernoulli, and T. Braun, “Receiver-based backbone construction and maintenance for wireless sensor or multi-hop networks,” in *Proc. of the 4th International Workshop on Mobile Ad-Hoc Networks*, Bern, Switzerland, 2007, pp. 409–420.

[36] M. Wälchli, S. Bissig, and T. Braun, “Intensity-based object localization and tracking with wireless sensors,” in *Workshop on Real-World Wireless Sensor Networks*, 2006.

[37] M. Wälchli, S. Bissig, M. Meer, and T. Braun, “Distributed event tracking and classification in wireless sensor networks,” *Journal of Internet Engineering (JIE)*, pp. 117 – 126, 2008.

[38] M. Wälchli, P. Skoczylas, M. Meer, and T. Braun, “Distributed event localization and tracking with wireless sensors,” in *Proc. of the 5th International Conference on Wired/Wireless Internet Communications*, Coimbra, Portugal, 2007, pp. 247 – 258.

[39] P. Wan, K. Alzoubi, and O. Frieder, “Distributed construction of connected dominating set in wireless ad hoc networks.” in *Proc. of 21st International Annual Joint Conference of the IEEE Computer and Communications Societies*, New York, NY, USA, 2002, pp. 141–149.

[40] J. Wu, M. Gao, and I. Stojmenovic, “On calculating power-aware connected dominating sets for efficient routing in ad hoc wireless networks,” in *Proc. of the 2001 International Conference on Parallel Processing*, Washington, DC, USA, 2001, pp. 346–356.

[41] J. Wu and H. Li, “On calculating connected dominating set for efficient routing in ad hoc wireless networks,” in *Proc. of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*, New York, NY, USA, 1999, pp. 7–14.

[42] D. Zhou, M. Sun, and T. Lai, “A timer-based protocol for connected dominating set construction in ieee 802.11 multihop mobile ad hoc networks,” in *Proc. of the The 2005 Symposium on Applications and the Internet*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 2–8.

[43] Y. Zou and K. Chakrabarty, “Sensor deployment and target localization in distributed sensor networks,” *Trans. on Embedded Computing Sys.*, vol. 3, no. 1, pp. 61–91, 2004.