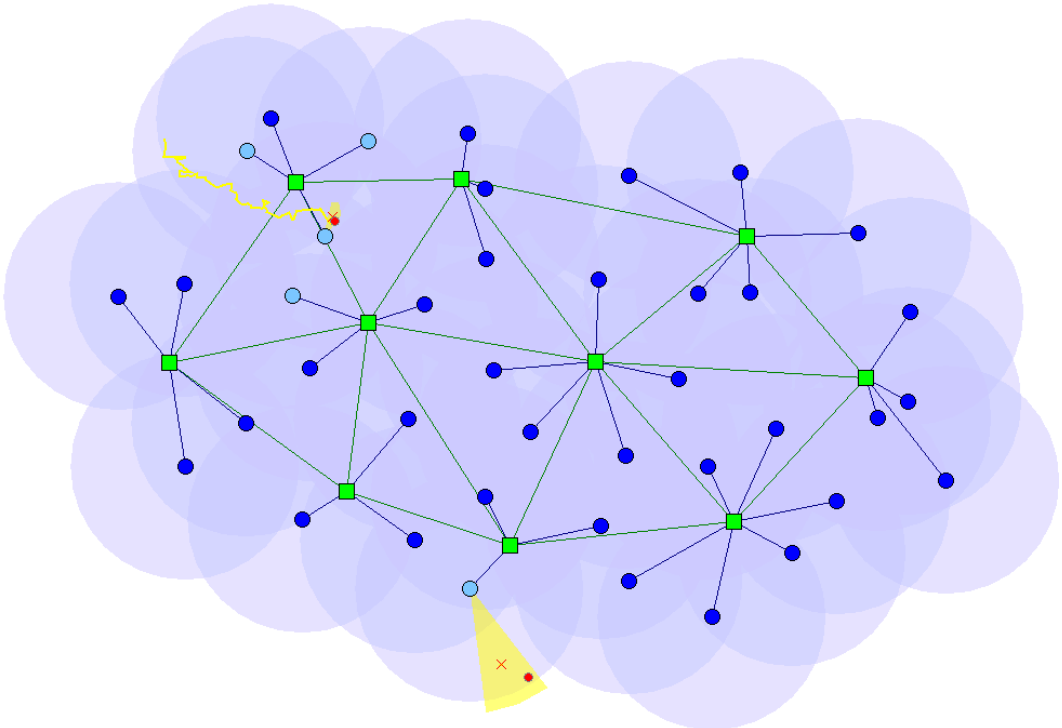
FOI-R--0500--SE
May 2002
ISSN 1650-1942
Scientific report

Mikael Brännström

An Agent Architecture for an Unattended Ground Sensor Network



Command and Control Systems
SE-581 11 Linköping

SWEDISH DEFENCE RESEARCH AGENCY

Command and Control Systems
P.O. Box 1165
SE-581 11 Linköping

FOI-R--0500--SE

May 2002

ISSN 1650-1942

Scientific report

Mikael Brännström

An Agent Architecture for an Unattended Ground Sensor Network

Issuing organization	Report number, ISRN	Report type
FOI – Swedish Defence Research Agency Command and Control Systems P.O. Box 1165 SE-581 11 Linköping	FOI-R--0500--SE	Scientific report
	Research area code 4. C4ISR	
	Month year May 2002	**Project no.** E7036
	Customers code 5. Commissioned Research	
	Sub area code 42 Surveillance Sensors	

Author/s (editor/s)
Mikael Brännström

Project manager

Approved by

Sponsoring agency

Scientifically and technically responsible

Report title
An Agent Architecture for an Unattended Ground Sensor Network

Abstract (not more than 200 words)
The technical evolution the last decade with the outcome of new small sensors has enabled the development of unattended ground sensor networks. The missing part is an architecture that enables autonomous detection, tracking and classification of targets in the covered area while minimizing communication and sensor usage. A sensor network architecture, with mobile track agents that moves through the network as the tracked target moves, has been developed and implemented. Dispatching of sensor data, fusion and power management are also handled by the agents. Simulated sensors produce the sensor data from targets in the simulation environment. To support different sensor types to be connected, a general abstraction level of uncertainty areas and sensor models has been introduced. The result is a general, robust sensor network system that can detect and track multiple targets and scales well when increasing the number of sensors and computational nodes in the network.

Keywords
agents, architecture, unattended ground sensor network, tracking

Further bibliographic information	Language English
Earlier edition published as Final Thesis report at Linköping University, reg no LiTH-IDA-Ex-02/21	

ISSN 1650-1942	Pages 45 p.
	Price acc. to pricelist

Utgivare	Rapportnummer, ISRN	Klassificering
Totalförsvarets Forskningsinstitut - FOI	FOI-R--0500--SE	Vetenskaplig rapport
Ledningssystem	**Forskningsområde**	
Box 1165	4. Spaning och ledning	
581 11 Linköping	**Månad, år**	**Projektnummer**
	Maj 2002	E7036
	Verksamhetsgren	
	5. Uppdragsfinansierad verksamhet	
	Delområde	
	42 Spaningssensorer	

Författare/redaktör	Projektledare
Mikael Brännström	
	Godkänd av
	Uppdragsgivare/kundbeteckning
	Tekniskt och/eller vetenskapligt ansvarig

Rapportens titel (i översättning)

En agentarkitektur för ett autonomt marksensornät

Sammanfattning (högst 200 ord)

Den tekniska evolutionen det senaste årtiondet som resulterat i nya, små och billiga sensorer har möjliggjort utveckling av autonoma marksensornät. Pusselbiten som saknas är en arkitektur som gör det möjligt för autonom detektion, spårning och klassning av objekt i det bevakade området samtidigt som nyttjande av sensorer och kommunikation minimeras. En arkitektur för ett autonomt marksensornät, med mobila spåragenter som förflyttar sig i nätverket alltefter som de spårade objekten rör sig, har utvecklats och implementerats. Transport av sensordata, datafusion och strömsparfunktioner hanteras också av agenter. Simulerade sensorer genererar sensordata utifrån objekt i simuleringsmiljön. För att stödja flera olika sensortyper har ett generellt abstraktionslager med osäkerhetsytor och sensormodeller skapats. Resultatet är ett generellt och robust autonomt marksensornät som kan upptäcka och spåra flera objekt samtidigt som nätet skalar upp när man ökar antalet sensorer och beräkningsnoder.

Nyckelord

agenter, arkitektur, autonomt marksensornät, målföljning

Övriga bibliografiska uppgifter	Språk Engelska

Tidigare utgåva publicerad som examensarbete vid Linköpings tekniska högskola, reg nr LiTH-IDA-Ex-02/21

ISSN 1650-1942	Antal sidor: 45 s.
Distribution enligt missiv	Pris: Enligt prislista

Table of Contents

Chapter 1 Introduction

The interest in unattended ground sensor networks (UGS) has increased the last few years. The reason for this is the technical evolution that has taken place the last decade resulting in new small, inexpensive and low-powered sensors. The Bluetooth concept for communication has also influenced the design of this type of network. Moreover, research on sensors and data fusion [1] has resulted in new algorithms and methods to this area. Other interesting aspects of unattended ground sensor networks are the various techniques that can be used to deploy the sensors and computational nodes, e.g. from different types of flying platforms, by grenades fired from various pieces of ordnance, or through manual deployment.

A UGS network must be robust to environmental noise as well as failure of sensors and computational nodes to be considered truly functional. An efficient activation/deactivation scheme for the sensors must be developed to increase the lifetime of the network. Furthermore, the UGS network must support different types of sensors and be scalable so that new sensors and nodes can be connected without reducing the performance of the network. Traditional architectures do not fulfil all these requirements and may be too cumbersome to maintain. New system architectures must thus be developed.

1.1 Problem Definition

A major problem when developing this kind of new large-scale systems is that the underlying hardware does not yet exist, at least not in a large number. The exact suite of sensors that will need to be used in an unattended ground sensor network is not known, and the current configuration might change rapidly. The system architecture for this sensor network should be general enough to handle different kinds of sensors, even future ones. The sensor network should be able to detect and track multiple targets in the covered area. These requirements and the fact that large-scale testing of the sensor network is required during the method development make simulation of these hardware components necessary.

The system architecture should function in distributed environments and scale well with an increasing number of sensors and tracks. The complexity of the problem points out the necessity for new models and techniques. The system architecture should be based on agents.

1.2 Aim of this Work

The aim of this work is to design an agent-based architecture for a UGS network as discussed in the problem definition. The focus is on the higher level of a UGS network, i.e. sensor-near fusion and network communication etc. are excluded.

A second objective is to implement a system and simulation environment that functions adequately using the proposed architecture, and as a sub-goal to look a little further into multiple targets and sensor activation/deactivation. Since no sensors are available these will be simulated.

1.3 Assumptions

Since this is a very open problem, some assumptions were made. Assumptions about the sensors as listed below:

1. they can determine their own location with some known accuracy;
2. an uncertainty area can be computed from any reported observation;
3. a signature, used for classification, can be computed from a reported observation.

The first assumption concerning the sensors also applies to the nodes, i.e. the nodes must also be able to determine their own locations. Some assumptions about the network that simplify the problem to some degree have also been made:

1. each node can communicate with any other node in the network;
2. there are no delays in the network.

1.4 Structure of the Report

This report is organized as follows: chapter 2 contains the background material needed for this report. This includes the concept of unattended ground sensor network, agents and related work. In chapter 3 the system architecture is described in detail starting with an overview, continuing with the network topology and the agent architecture. In chapter 4 the system implementation and the simulation environment are covered. In chapter 5 the tracking of multiple targets and the sensor activation algorithm for tracking are discussed. The last chapter of the report, chapter 6, contains some concluding remarks including a summary and a discussion about future work. Finally there is an appendix with a short user's manual.

Chapter 2 Background

2.1 Unattended Ground Sensor Network

Unattended ground sensor networks generally consist of many small, spatially dispersed sensors. A network between the sensors is established, with short-range (ten to hundreds of metres) wireless communication. Often taken into account is that the sensors can fail or new sensors are added, and that communication links are not reliable. Robustness is thus often a keyword when talking about sensor networks. Unattended means that the network is not being looked after regularly. When the sensors have been deployed they are on their own. In order for the sensors to cooperate, fusion of sensor data must be available, thus cooperatively they accomplish tasks and provide capabilities greater than the sum of the individual parts. [2][3][4][5]

When talking about unattended ground sensor networks in the military context, the main objective of the network is to detect, track and classify targets in the covered area. Although research began with the aim of military applications, civilian applications also exist. Sensor networks can be used in case of chemical or radiation accidents or to observe the spreading of forest fire. In this report the focus is, however, on military applications where the goal is to detect, track and classify targets. [3][5]

The sensor network consists mainly of its sensors. A sensor is defined as:

"*anything, such as a photoelectric cell, that receives a signal or stimulus and responds to it*"[1]

A categorization often used is to divide the sensors into active and passive sensors. Passive sensors do not reveal their location by radiation when receiving information. These sensors are therefore more attractive for use in a sensor network, since sensors that have revealed their presence become targets to countermeasures such as jamming of communication or information gathering, annihilation and tampering. Active sensors have generally higher energy consumption, which in turn requires larger batteries or more complex sleep-states and wake-up logic.

[1] William Collins Sons & Co. Ltd., Collins English Dictionary, 1986, ISBN 0-00-433134-x

Some sensors suitable for sensor networks [6] are:

Passive sensors:

Acoustic:	microphone, geophone, pressure, strain
Optical:	image sensors, PIR-sensors
Field sensors:	magnetic, field strength receivers, GPS, mm-wave
Radiation detectors:	neutron radiation, gamma radiation
Chemical:	gas sensors

Active sensors:

Optical:	laser radar, image sensors (illuminating), IR-sensor
Microwave:	radar
Magnetic field sensor:	magnetometers
Multi-point sensors:	mechanical, optical

2.2 Agents

Increasingly many computer systems are being viewed in terms of agents. Not only the AI community is talking about agents, but also researchers in mainstream computer science, as well as those working in data communication and concurrent system research, discuss this. [7]

2.2.1 What are Agents?

Although agents are widely discussed, there does not exist an agreement of what agents are. The definition depends on the area that agents are applied to. Wooldridge and Jennings [7] point out some properties of an agent:

- Autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;
- Social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language;
- Reactivity: agents perceive their environment and respond in timely fashion to changes that occur in it;
- Pro-activeness: agents do not simply act in response to their environment; they are able to exhibit goal-directed behaviour by taking the initiative.

Wooldridge also presents a definition of an agent [8]:

"An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives."

This is the definition that will be used in this report.

2.2.2 Motivation

Building high quality software for complex real-world applications is difficult. This also applies to an unattended ground sensor network. Although object-oriented programming helps us to model and design a system, the building blocks of today are too finely grained. Jennings argues that: “agent-oriented approaches can significantly enhance our ability to model, design and build complex (distributed) software systems” [9].

2.3 Related Work

The research in this area is extensive and many projects about unattended sensor networks exist. A few are listed below.

The Sensor Information Technology program (SensIT) [5], sponsored by the Information Technology Office (ITO) at the Defence Advanced Research Projects Agency (DARPA), is developing software for networks of distributed microsensors. A microsensor device will have multiple onboard sensors, such as acoustic, seismic, infrared and magnetic sensors, embedded processing and storage, short-range wireless links and positioning capabilities. They also mention the “smart dust” concept, with microsensors on the order of square millimetres or square centimetres in size. SensIT researchers are also testing their software in the field with the assistance of the US Marine Corps.

At the department of Computer Science and Engineering at the Auburn University, research is ongoing on highly mobile sensors [2]. They are working with a sensor network that offers mainly three distributed services; lookup service, composition service and dynamic adaptation service. These services provide support for dynamic information dissemination and fusion that adapts to incremental addition and removal of sensor nodes, device failures and degradation, etc.

Hardware system architecture for networked sensors has been developed at the department of Electrical Engineering and Computer Sciences at the University of California, Berkeley [4]. A small device with a tiny operating system, which fits into only 178 bytes of memory, has been created to lay the groundwork for future architectural advances.

At the department of Electrical and Computer Engineering at the University of Wisconsin, Madison, research is being done about a framework for collaborative signal processing in distributed sensor networks [10]. There people look into tracking of multiple targets, where classification of targets is done with the help of neural networks.

Chapter 3 System Architecture

This chapter describes the architecture of the system. It also discusses the design considerations made.

3.1 Overview

The design goal is to create a system architecture that is general enough to support many different kinds of sensors, while being distributed and scalable. To deal with this complexity, the system architecture is based on agents, see section 2.2.

The sensor network being simulated consists of computational nodes, sensors and a communication link on which observations to an external user or system are sent. The nodes and sensors are aware of their (at least relative) positions. More than one node may have the capability of communicating with external users or systems. The communication link is the logical description of this capability. The computational nodes, or just nodes for short, have computational power for fusion and hold an environment for the agents. The sensors are connected to the nodes that are interconnected via a network. Figure 1 shows an example of what the sensor network may look like.

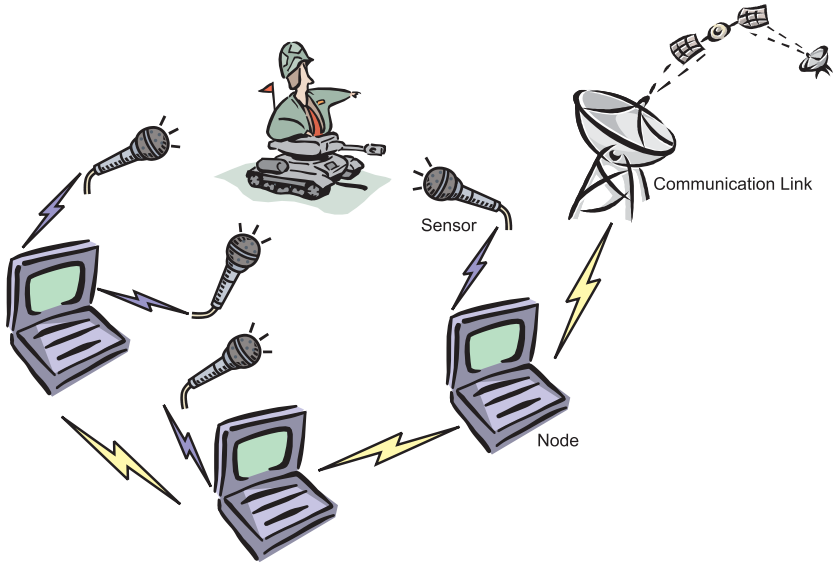


Figure 1 *Sensor network.*

3.2 Network Topology

To make it possible for agents to collaborate there must exist an environment in which they can find each other. The hardware environment consists of the nodes. These nodes are connected via a network, where each node can communicate with any other node, directly or indirectly.

To make the network scale up when increasing the number of nodes, each node only has direct contact with its closest neighbouring nodes. This means that, on the average, each node is connected to about six other nodes, if the nodes are evenly distributed. This number rarely exceeds 16 in practice [11]. To calculate which nodes that should be connected to each other, Delaunay triangulation (described below) is used. Each sensor is connected to the geographically closest node, determined through Delaunay triangulation. An example of the network topology is shown in Figure 2.

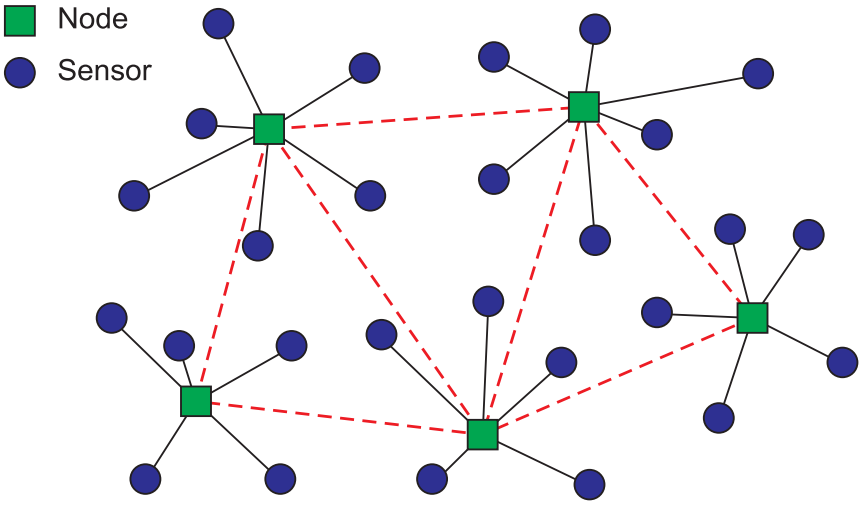


Figure 2 *Topology of the network.*

3.2.1 Delaunay Triangulation

Delaunay triangulation is best described by its dual, the Voronoi diagram. The Voronoi diagram of a set $S = \{p_1, p_2, ..., p_n\}$ of points in the plane, called sites, is a partitioning of n convex polygons, one per site. Each Voronoi region V_i contains all points in the plane closer to p_i than to any other site. The Delaunay triangulation is obtained by adding a line segment between each pair of sites in S, whose Voronoi regions share an edge. An example of a Delaunay triangulation and a Voronoi diagram is shown in Figure 3.

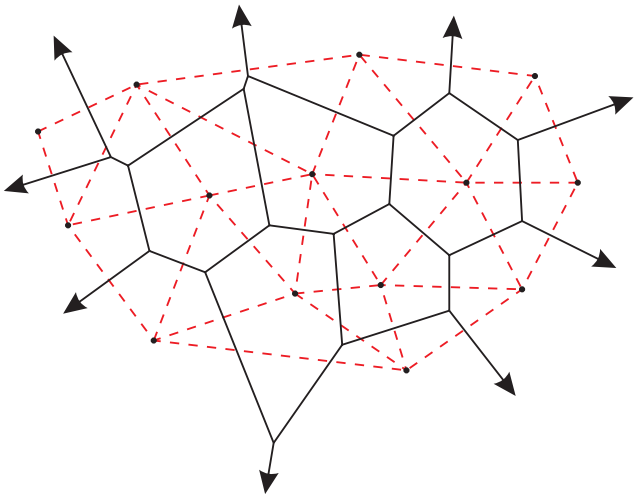


Figure 3 *Delaunay triangulation (dashed) and its dual, the Voronoi diagram (solid).*

A commonly used algorithm for this problem is the Guibas-Stolfi algorithm [12]. Both the Delaunay triangulation and the Voronoi diagram are represented by the data structure used in the algorithm. The time complexity of the algorithm is $O(n \log n)$ for a diagram of n sites, and $O(n)$ for each additional site to be inserted. Although the standard triangulation algorithms, including Guibas-Stolfi, do not support deletion of sites there exist methods for doing so [11]. The algorithm used here is an incremental version of the Guibas-Stolfi algorithm. This algorithm has been chosen because of its simplicity and because it is only slightly slower than the fastest algorithm, the Fortune algorithm [13].

3.3 Agent Architecture

This system is based on a number of different agents that all have different tasks, and the same interface for activation and communication. The common interface is described, followed by a more detailed description of the different agents and their tasks. Finally the collaboration between the agents is discussed.

3.3.1 Agent Design

All agents have a common interface. This interface contains methods for communication, notification of new cycle from the framework and a method telling whether the agent is dead or not, see Table 1. When an agent has marked itself as dead, it is removed by the agent environment.

Table 1 *The agent interface.*

Method name	Return type	Description
communicate(message)	Response	Sends a *message* to the agent. The agent replies with a *response*.
isDead()	boolean	Checks if the agent is dead.
newCycle(cycle)	void	Informs the agent about the new *cycle*.

The agents can send messages to each other using the communicate method. A message consists of a message type, see Table 2, and an optional payload. The called agent replies with a response containing a response type and an optional payload. The response type can be one of the following:

- OK – The message was accepted.
- Negative – The message was rejected.
- Reply – The message will be replied to with another message.
- Unknown message – The message was not understood.

Table 2 *The message types that are communicated between agents.*

Message type	Description
`MSG_GET_NEIGHBOURS`	Get a vector of the neighbouring nodes.
`MSG_GET_DETECTABLE_AREA`	Get the detectable area of a sensor.
`MSG_GET_COVERAGE`	Get the sensor coverage at a position.
`MSG_GET_REFERENCE`	Get a sensor reference from a sensor agent.
`MSG_GET_SENSORS`	Get a vector of sensors that are connected to the local node.
`MSG_SENSOR_DATA`	Inform about new sensor data.
`MSG_CHECK_SENSOR_DATA`	Check the sensor data match.
`MSG_SPAWN_TRACK_AGENT`	Spawn a new track agent from a track agent state. Used for track agent mobility.
`MSG_CLAIM_USAGE`	Claim usage of a sensor.
`MSG_SENSOR_DATA_COLLISION`	Inform that sensor data collision has occurred.
`MSG_CONNECT_NODE`	Connect a neighbouring node to this node.
`MSG_CONNECT_SENSOR`	Connect a sensor to this node.

3.3.2 Agent Types

To solve the task of detecting and tracking targets, a number of agent types have been designed. Different tasks have been assigned to the different types. One goal that they all have in common is to minimize the amount of power used.

This means that they should minimize communication, both between nodes and between sensors and nodes, and minimize the usage of the sensors. This goal often contradicts other agent tasks. For example the track agent's goal to track a target with high accuracy implies that many sensors should be used, which contradicts the goal of minimizing sensor usage.

Node Agent

Each node is equipped with a node agent. The responsibility of the node agent is to keep track of the neighbouring nodes, i.e. the nodes that are geographically closest to the current node. The node agent also keeps track of the node coverage area, which is the combined detectable area of all local (directly connected) sensors' detectable areas. The node agent is therefore able to answer how many sensors that can be used to detect an object at a given position.

Another responsibility of the node agent is to support track agent movement between nodes. When a track agent needs to move from a node to another, it asks the node agent at the target node to create a new track agent using a supplied track agent state. This track agent state contains all necessary information to create a new copy of the track agent.

Dispatch Agent

A dispatch agent exists at each node and it handles sensor data messages received from a sensor agent. When receiving sensor data from the sensor agent, the dispatch agent will do the following (in the specified order):

1. Check local (at the same node) track agents to see if the sensor data matches any of the track agents' tracks. If more than one track agent is interested in the particular sensor data, then all the interested track agents will get a sensor data conflict message containing references to all the other agents and the sensor data id.
2. If no interested track agent was found in the previous step, the dispatch agent asks the neighbouring dispatch agents to check their local track agents. This is done in the same manner as in step 1.
3. If no neighbouring dispatch agent could find an interested track agent, then this sensor data corresponds to a new target and a new track agent is created.

Sensor Agent

When a sensor is connected to a node, a new sensor agent is created. This sensor agent will, from that point on, handle all communication with that particular sensor. The sensor agent handles activation and deactivation of the sensor.

Activation of a sensor is made either upon request by another agent or fired by a timeout (caused by a timer) in the sensor agent to check if there are any targets

in the area. When the sensor is activated it will listen and send sensor data as soon as it detects something. If no agent wants to use the sensor it will be deactivated and the timer will be restarted. The timer thus makes sure that the sensor is activated regularly in order to be able to detect new targets.

When the sensor agent receives sensor data from the sensor it will compute an uncertainty area of the observation based upon the sensor model and the position of the sensor. Here the accuracy of the sensor position is also considered. Then this uncertainty area is attached to the sensor data, and the sensor data is passed on to the dispatch agent that makes sure that the sensor data is handled in a correct way.

Track Agent

For each known target, a track agent exists, whose task is to track that target. The track agent should also try to minimize inter-node communication. This means that the track agent should reside at the node nearest to the sensors it intends to use.

Whenever a new target is detected, a new track agent is created. The track agent is from that point dependent on the arrival of new sensor data. If a track agent does not receive new sensor data within a certain number of cycles, the target is considered lost and the agent marks itself as dead. The node framework will then remove the agent at the end of the cycle.

This means that the track agent has to ensure that it will receive new sensor data in each cycle by claiming the use of the sensors that can detect the target. When there is only one sensor that can detect the target, the choice is easy. When there are several sensors that can detect the target, the choice of which sensor or sensors that should be used is more difficult. Solutions to this problem are discussed further in section 3.5.2.

When the track agent tracks a target it keeps an uncertainty area of the target position, the target position and a set of possible target types. The actual tracking algorithm is described in section 3.4.

When many nodes exist the track agent also has the option to move to another node. The scheme for movement of the agent is quite simple. The track agent will move from the current node if, and only if it finds a neighbouring node that has better sensor coverage at the target position. This information is retrieved from the node agent in this node, and from the node agents in the neighbouring nodes. It is important not to move between nodes too often because the movement in itself also introduces a communication cost.

3.3.3 Agent Collaboration

In order for the agents to find each other, each agent has a reference to the node where it resides. At each node there is a reference to each agent in that node. This way all agents can at least find any other agent in the current node. Since the node agent has references to the neighbouring nodes, it is also possible for an agent to find and communicate with any agent in the entire network of nodes.

The agent collaboration chain begins when a sensor agent receives new sensor data from a sensor. The sensor agent then processes the sensor data and passes it over to the local dispatch agent, see Figure 4. The dispatch agent will then ask the local track agents if they are interested in that particular sensor data, and if this is not the case the neighbouring dispatch agents will be asked the same question. Assume that the sensor data eventually is sent to an interested track agent. This track agent will then consume that sensor data, and as a result it will have a somewhat better perception of the track. If no track agent is interested in the sensor data, the dispatch agent creates a new track agent.

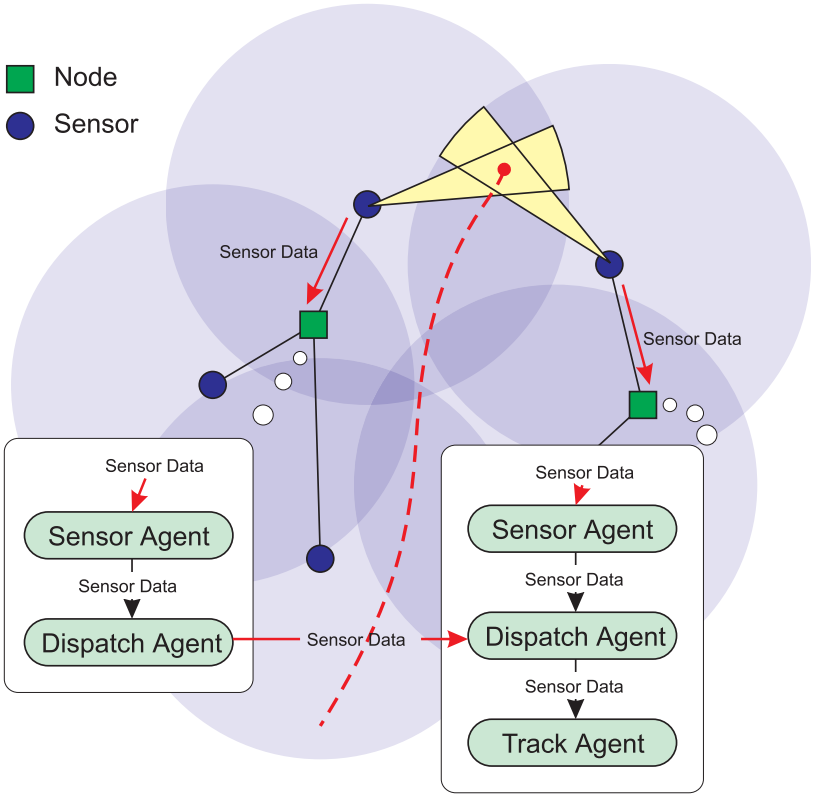


Figure 4 *Agent collaboration when receiving sensor data from sensors connected to different nodes.*

To keep track of the target, the track agent must receive new sensor data in the next cycle. After choosing the sensors to use, the track agent sends a “claim sensor” message to the corresponding sensor agent, which makes sure that the sensor is active in the next cycle.

The scheme for movement of the track agent is a result of collaboration between the node agent, the sensor agent and the track agent. The node agent gathers the detectable areas of the sensors from the local sensor agents. Then the track agent asks the node agent about the number of sensors that have coverage at a certain position. This way the track agent can easily check which node that has the best coverage at the target position, i.e. the node that has the most sensors with coverage at the target position. The probability that the majority of the sensor usage is local is maximized, by choosing a node with the highest sensor coverage at the target position.

3.4 Tracking Algorithm

The track agent keeps track of the uncertainty area of the target position, a set of target types and the target position. Currently the target position is computed as the mass centre of the uncertainty area, but one can easily imagine an algorithm that tries to fit a spline curve to the previous uncertainty areas and positions. This curve combined with an estimate of the target speed could then be used to predict the next position of the target.

The track agent regularly receives refined sensor data. Before the track agent receives the sensor data the sensor agent refines the sensor data by taking the uncertainty of the sensor position and the observation uncertainty into account. The refined sensor data contains an uncertainty area of the observation. The uncertainty area of the observation is then combined with the current knowledge of the target uncertainty area. The motivation for using uncertainty areas will be discussed before the area fusion algorithm is explained further.

3.4.1 Motivation for the Use of Uncertainty Areas

Since many different types of sensors should be supported, either fusion algorithms that can be applied to all the different combinations of observation types must be developed, or a common abstraction level (at which fusion is done) of an observation must be introduced.

The first method requires a great deal of development and introduces unnecessary complexity. The choice fell on the second method, i.e. introduction of a common abstraction level of an observation. This method requires implementation of only one fusion algorithm. This algorithm may not be as efficient as using specialized algorithms for different combinations of observation types, but the simplicity and the reduction of development time makes this method far more attractive. Using a simple algorithm without a lot of special cases should also make a positive impact when considering bugs and robustness.

Since the simulated world is two-dimensional, the set of positions of a target is always an area, possibly consisting of multiple sub-areas. Most observations can

easily be transformed into an uncertainty area. For example, a direction will become a pie-shaped area with, in practice, a finite radius.

3.4.2 Sensor Data Fusion

When the track agent receives sensor data it checks that the sensor data matches the target it tracks. First of all, the uncertainty area of the observation must overlap the current perception of the target uncertainty area. Secondly, the classification set of the target is compared to the classification set of the observation. Finally, the sensor model of the source of the observation is checked to see if it is theoretically possible that the source could detect this target type anywhere in the target uncertainty area. If all these conditions are met, the sensor data is accepted.

Collisions of matched sensor data will occur if there are multiple targets that are being tracked. There are two types of collisions: sensor collisions and track collisions. A sensor will send at most one observation of every target, i.e. it will never send two observations of the same target during the same cycle. There is thus a sensor collision if a track agent receives multiple matching sensor data from a single sensor. There is a track collision if more than one track agent accepts a single sensor data. These two collision types will often occur simultaneously and they indicate that an erroneous match is likely. Many of these collisions can be avoided by simply applying sensor data in the right order. If sensor data without collisions are applied first, the perception of the target is improved and some of the sensor data that was accepted earlier may now be rejected. This way some of the collisions can easily be resolved. No further collision resolving techniques are used in the current implementation. Sensor data with collisions are not used because of the risk of making an erroneous match.

Using this restrictive method, when accepting sensor data, can cause the track agent to starve. Therefore, the sensor activation algorithm used to reduce the number of sensors that are activated is not used when a collision has been detected. This means that all sensors that can detect the target will be activated to minimize the risk for starvation.

When using sensor data that has been accepted, the new target uncertainty area is the intersection of the old target uncertainty area and the sensor data uncertainty area. The same is done for the target classification set. The new target classification set is the intersection of the old classification set and the classification set of the sensor data. Note that to accept the sensor data both the uncertainty areas and classification sets must overlap.

The target uncertainty area is expanded in each cycle. This expansion is caused by the mobility of the target. The amount of expansion is the maximum distance

the target type or types can move during one cycle. This means that in order to keep the uncertainty area size at a somewhat constant level, new sensor data must arrive regularly.

3.4.3 Target Classification

In order to separate two (or more) nearby tracks from each other, other measures than position are needed. Classification of detected targets into one or more possible target types is often a wanted function of a UGS network and is also such a measure.

In this implementation a very simple target classification technique is used, that gives a measure of how likely a particular classification is. The most likely target types forms a set, which is called the target classification set. This set can then be used to separate sensor data from each other if the sensor data signature differs enough.

3.5 Sensor Activation Algorithm

There are two aspects of sensor activation in this system. One aspect is detection of new targets, and is controlled by the sensor agents. The other is the tracking of known targets, and is thus controlled by the track agents.

3.5.1 Sensor Activation Algorithm for Detection

The goal of the sensor activation algorithm for target detection is to minimize the total number of activation cycles while guaranteeing that a detectable target will be detected within a specified number of cycles. This means that the sensors should periodically be activated to check if there are any targets within their detectable range.

A concept of “edge sensors” has been proposed [14] to solve this problem. It is here assumed that the detectable areas of the sensors overlap and the outer edge sensors can be found. Then only these edge sensors are active and are thus forming a “trip wire” that alerts the other sensors when a target has entered the sensor network. Since the edge sensors will be active more often than the other sensors, they will lose power faster and the area covered by the sensor network will shrink as the former edge sensors die.

The major drawback of this interesting concept is the lack of robustness. If a target sneaks through an edge sensor, it is safe from detection while inside the “fence”. An edge sensor might be located in a “shadow” in which it will never get any observations, or be broken without knowing it. When dealing with different types of sensors and targets, it can be difficult to find these edge sensors. A candidate edge sensor that can detect one target type may not be able to detect another target type. All this can of course be accounted for, with multiple layers of edge sensors. This concept comes best in use when dealing

with many sensors (a couple of hundreds) because of its good scaling property. The number of edge sensors is proportional to the square root of the number of sensors in the optimal case when the sensors are evenly spread out in a circle.

The method implemented has a reasonable degree of efficiency and is simple and robust. The method is simply to activate each sensor regularly. If a sensor has not been used during a number of cycles it is activated to check if there are any targets to be detected. This means that areas with high sensor coverage will be observed more frequently than really needed. It is at this point that the method can be made more efficient.

3.5.2 Sensor Activation Algorithm for Tracking

The purpose of the sensor activation algorithm for tracking is to minimize the use of the sensors, i.e. minimize the energy consumption, while tracking a target. The exact meaning of this is not clear. Nevertheless there are ways to measure the quality of a track perception. The size of the uncertainty area of the target could be used for this purpose. There does not either exist a given goal for an efficient algorithm. One could say that the goal is to minimize the use of sensors while keeping the uncertainty area of a target below a certain level. Since a certain level of accuracy cannot be guaranteed, this definition cannot be used without modification. Another possibility is to look at the relative gain: How much is the expected uncertainty area reduced with respect to the number of sensors used? This approach is used here and described further below.

Each track agent activates the sensors it needs without considering other track agents. It starts with a set of sensors that can detect the target in focus. The best sensor combinations, which reduce the target uncertainty area the most, are computed for 1, 2, … N sensors, where N is the number of sensors in the starting set. For each sensor that is added, the expected target uncertainty area must shrink a certain percentage. The largest sensor combination that fulfils this requirement is chosen. If no combination fulfils the requirement then the best single sensor is chosen. This way, at least one sensor is chosen and more sensors are added if they improve the result significantly.

Chapter 4 System Implementation and Simulation Environment

This chapter describes the simulation environment and the implemented system. A limited user's guide of the system can be found in Appendix A.

4.1 Overview

The system and the simulation environment are implemented in Java and consist of three programs: the simulation server, the communication link and the node. Each of these communicates via RMI (Remote Method Invocation) and can therefore be distributed over a number of interconnected computers. In a running simulation there is one instance of the simulation server, one instance of the communication link and one or more instances of the node program, see Figure 5. There also exist tools for creating scenario descriptions and for visualization of the simulations. These tools are described in Appendix A.

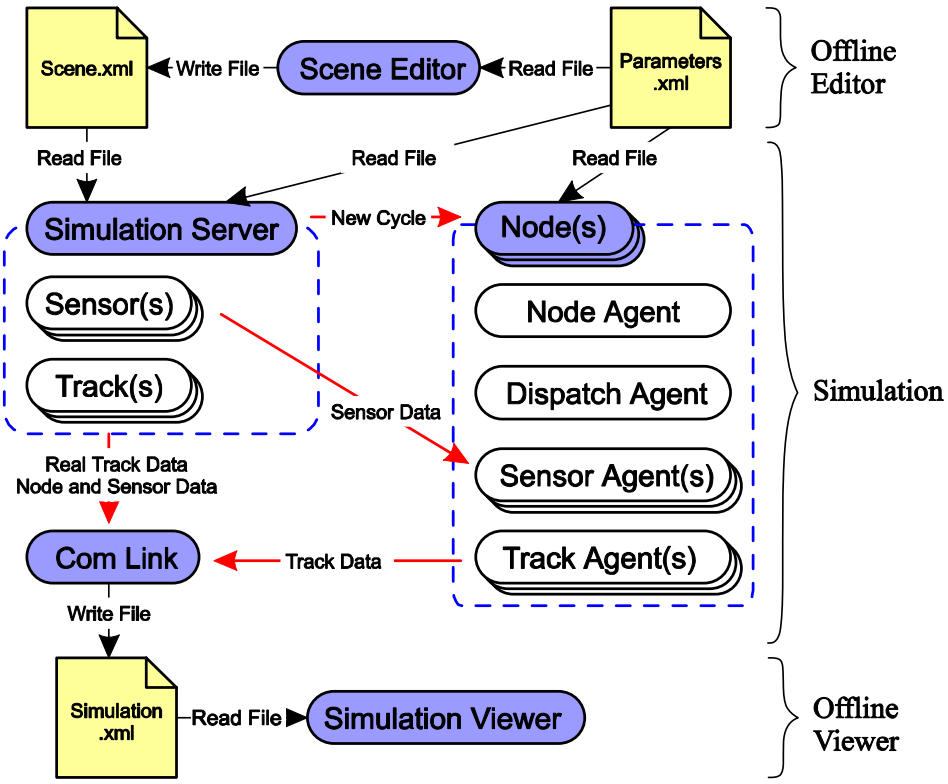


Figure 5 *Overview of the system.*

The input to the simulation consists of two XML files: the parameter file and the scene file. The parameter file contains the sensor models, target models and environment parameters such as temperature, humidity etc. The scene file describes a specific scene and contains sensors, nodes and tracks. Since the scene file contains the number of nodes, it affects how many instances of the node program needed.

The output of the simulation is stored in the simulation file, also in XML. It contains both observed and real tracks, and information of the sensors and the nodes in the simulation. These files are described further in section 4.2.

4.2 File Formats

As already stated, the environment requires two input files: the parameter file and the scene file. Furthermore the output of the system is stored in the simulation file. The format and content of these files are described below.

4.2.1 Parameter File

The parameter file is taken as input to the simulation server and the node. It contains information about all known sensor and target models, as well as environmental information. The parameter file has the following structure:

```
<parameters
cycle_time="time in seconds"
sound_frequences="frequences, f1 f2 ... fN, in Hertz"
temperature="temperature in degrees Celsius"
relative_humidity="humidity in percent"
max_acoustic_noise="intensity in W/m">

  <sensormodels>
    <sensormodel
    type="unique name"
    class="Java class name"
    self_localisation_accuracy="accuracy in meters">
      <acoustic_args
      min_frequency = "minimum frequency in Hertz"
      max_frequency = "maximum frequency in Hertz"
      intensity_threshold = "intensity in Watt"
      low_frequency_angle = "error angle in degrees"
      high_frequency_angle = "error angle in degrees"/>
    </sensormodel>
    ...
  </sensormodels>

  <targetmodels>
    <targetmodel
    type="unique name"
    max_speed="maximum speed in m/s"
    acoustic_signature="power, I1 I2 ... IN, in W"/>
    ...
  </targetmodels>
</parameters>
```

4.2.2 Scene File

The scene file describes a particular scene to be simulated and is only provided to the simulation server. It contains information about tracks, such as position of the target at each cycle, and the type of the target. The scene file also contains information about all the nodes and sensors in the scene. The scene file has the following structure:

```
<scene
start_cycle="start cycle"
duration="number of cycles">

  <nodes>
    <node
    x="x-coordinate in meters"
    y="y-coordinate in meters"/>
    ...
  </nodes>

  <sensors>
    <sensor
    type="sensor type, specified in the parameter file"
    class="Java class name"
    x="x-coordinate in meters"
    y="y-coordinate in meters"/>
    ...
  </sensors>

  <tracks>
    <track
    id="unique name"
    type="target type, specified in the parameter file"
    interpolation="interpolation between the points">
      <point
      x="x-coordinate in meters"
      y="y-coordinate in meters"
      cycle="timestamp for the point"/>
      ...
    </track>
    ...
  </tracks>
</scene>
```

4.2.3 Simulation File

The output of a simulation is stored in the simulation file. It is produced by the communication link, which receives information from the nodes and the simulation server. The simulation file contains information about the real and observed tracks. The file also contains information about the nodes and sensors used in the simulation. The simulation file has the following structure:

```
<simulation
start_cycle="start cycle"
duration="number of cycles">

  <nodes>
    <node
    id="unique name"
    x="x-coordinate in meters"
    y="y-coordinate in meters"/>
    ...
  </nodes>

  <sensors>
    <sensor
    id="unique name"
    type="sensor type, specified in the parameter file"
    x="known x-coordinate in meters"
    y="known y-coordinate in meters"
    real_x="real x-coordinate in meters"
    real_y="real y-coordinate in meters"
    active_cycles="cycles when the sensor was active"
    />
    ...
  </sensors>

  <tracks>
    <track
    id="unique name">
      <obs
      x="x-coordinate in meters"
      y="y-coordinate in meters"
      cycle="timestamp for the observation">
        <polygon>
          <point
          x="x-coordinate in meters"
          y="y-coordinate in meters"/>
          ...
        </polygon>
      </obs>
      ...
    </track>
    ...
    <real_track
    id="unique name"
    type="target type, specified in the parameter file">
      <fact
      x="x-coordinate in meters"
      y="y-coordinate in meters"
      cycle="timestamp of the fact"/>
      ...
    </real_track>
    ...
  </tracks>
</simulation>
```

4.3 Simulation

The simulator consists of the framework that drives the simulation, the tracks and the sensors that simulate the observations of the tracks. The simulated time is discrete; each cycle is 100 ms long. The coordinate system is two-dimensional. The framework starts reading the input files, sets up the scene and waits for the communication link and a number of nodes to connect. During the connection phase the sensors are spread out among the nodes, where each sensor is connected to the nearest node. The nodes are given a reference to the communication link and are informed about other nodes. Simulation begins when all necessary programs have been connected.

The simulation is actually an iteration of all the cycles. To facilitate synchronization, each cycle is divided into a number of phases see Figure 6. Each node is notified when a new cycle or phase is started. A new phase or cycle is started only when all nodes are finished with the current phase. This way, the phases function as synchronization barriers.

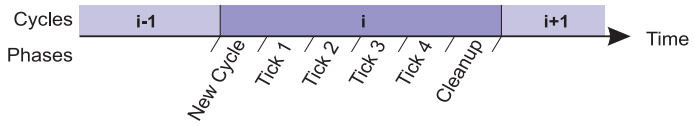


Figure 6 *The phases of a cycle.*

During the sense phase, each target is moved one step and then each sensor is given a chance to sense the targets. Whether a sensor reports a track observation by sending sensor data to the sensor listener or not, depends on the particular sensor implementation. Generally the sensor will report an observation if the sensor is activated and the sensor can detect the track according to the sensor model. Nothing is done on the server side of the other phases, except for notifying the nodes.

A sensor suite consists of implementations of a sensor model, a sensor, sensor data and a signature. The sensor model contains a model of sensor type and can compute various information such as detectable area, uncertainty area of a piece of sensor data and so on. The sensor itself creates sensor data from a target, using its position and type, and checking the constraints in the sensor model. The sensor data contains the actual observation, such as an angle or distance to the target, while the signature can be used for classification of the target. At present only an acoustic sensor suite is implemented.

4.3.1 Acoustic Data Model

Simulation of acoustic data is based on the fact that each periodic signal can be separated into a number (maybe infinite) of sinus-shaped signals with different

frequencies. Since all machines emit more or less periodic sounds, one can say that they in fact emit a spectrum of intensities at a number of frequencies.

This spectrum is here called the acoustic signature. This signature has operations for checking equality between different signatures, transforming and filtering the signature. The acoustic sensor is a triplet of microphones that detects the direction to the sound source. The acoustic sensor data contains an angle and an acoustic signature.

Outdoor Sound Propagation

Outdoor sound propagation [15] is affected by many mechanisms, including:

- source geometry and type (point, line, coherent, incoherent);
- meteorological conditions (wind and temperature variations, atmospheric turbulence);
- atmospheric absorption of sound;
- terrain type and contour (ground absorption of sound, reflection);
- obstructions (buildings, barriers, vegetation, etc.).

The targets are treated as point sources and only the atmospheric absorption of sound is considered. To consider other effects, knowledge of the terrain would be required. The terrain is thus considered to be flat.

The mechanisms behind atmospheric absorption have been extensively studied, empirically quantified and codified into an international standard for calculation: ANSI Standard S1-26:1995 or ISO 9613-1:1996.

For a standard pressure of one atmosphere, the absorption A_{abs} (dB/m) can be calculated as a function of frequency f (Hz), temperature T (degrees Kelvin) and relative humidity h (%) by:

$$A_{abs} = 8.69 \cdot f^2 \cdot \left(1.84 \cdot 10^{-11} T_{rel}^{1/2} + T_{rel}^{-5/2} \left(0.01275 \frac{e^{-2239.1/T}}{F_{r,O} + f^2 / F_{r,O}} + 0.1068 \frac{e^{-3352/T}}{F_{r,N} + f^2 / F_{r,N}} \right) \right)$$

$$F_{r,O} = 24 + 4.04 \cdot 10^4 h \frac{0.02 + h}{0.391 + h} \qquad \text{Oxygen relaxation frequency (Hz)}$$

$$F_{r,N} = T_{rel}^{-1/2} \left(9 + 280 h e^{-4.17 T_{rel}^{-1/3} - 1} \right) \qquad \text{Nitrogen relaxation frequency (Hz)}$$

$$T_{rel} = \frac{T}{293.15}$$ Relative temperature

The absorption coefficients need only be calculated once for each frequency under the assumption that the temperature and the relative humidity remain constant. The intensity I (watts/m^2) received from a sound source can be calculated as a function of the sound power W (watts), atmospheric absorption A_{abs} (dB/m) and the distance d (m) to the source by:

$$I = \frac{W}{4\pi d^2} \cdot 10^{-0.1 A_{abs} d}$$

4.3.2 Acoustic Sensor Model

The acoustic sensor is, as mentioned, a combination of three microphones that can detect the direction to the sound source. To model when a sensor can detect a sound or a signature, it is assumed that the sensor has a lower and an upper boundary of which frequencies that can be sensed. Furthermore, the intensity of the sound must be higher than a certain threshold to filter out background noise.

The accuracy of the measured direction depends on the frequency since the number of periods that the sound differs at the different microphones is used to calculate the direction. This means that the direction to a sound source with high frequency has better accuracy than if the sound had low frequency. To take this effect into account, two error angles can be specified: one for high frequencies and one for low frequencies. These two frequencies refer to the minimum and maximum frequency that the sensor can detect. In between, linear interpolation is used to compute the error angle.

The detectable area of the sensor is computed by determining the maximum detection distance of all target models. The maximum detection distance of a target is simply the maximum distance when the intensity of its transformed and filtered acoustic signature is above the intensity threshold of the sensor. This is solved using the Newton-Raphson method. This detectable area is the union of the individual detectable areas for each target type, and is thus only used as an upper bound, i.e. the area where the sensor might detect targets.

Chapter 5 Evaluation

In the evaluation of the system the sensor activation algorithm for tracking was evaluated and compared to some other strategies. The overall behaviour was tested when tracking multiple targets. Finally the performance was tested when adding more nodes to the sensor network.

5.1 Sensor Activation Algorithm for Tracking

During the development of this unattended ground sensor system, a more thorough evaluation of the sensor activation algorithm for tracking (section 3.5.2) has been done. Three different configurations of this algorithm have been compared with three other strategies. The aspects that have been examined are the size of the target uncertainty area and the sensor usage of each strategy.

5.1.1 Evaluated Strategies

The sensor activation algorithm for tracking, fully described in section 3.5.2, uses area decrement as basis for choosing how many sensors that are going to be used. To add a sensor, to the chosen set of sensors, the expected uncertainty area size must be decreased by a certain percentage. This strategy is therefore here called the area decrement strategy. This algorithm is run in three different configurations since it is not clear which parameter values to choose.

The different strategies that were compared are:

- Area decrement 30% – add an extra sensor if it will reduce the area by at least 30 percent;
- Area decrement 20% – add an extra sensor if it will reduce the area by at least 20 percent;
- Area decrement 10% – add an extra sensor if it will reduce the area by at least 10 percent;
- Best-combo-2 – choose the best combination of two sensors;
- Random-2 – choose two sensors at random;
- All sensors – choose all sensors.

In all of these strategies, sensors are selected from the set of sensors that can detect the target. When there is only one sensor that can detect the target, there need be no choice made and all strategies will choose that particular sensor.

5.1.2 Scene Configuration

Three different scenes have been used in evaluating the algorithms. The scenes consist of one node, one track and 20, 30 respectively 40 sensors that are randomly distributed within an ellipse, see Figure 7. All scenes are 200 cycles long and the track motion is the same in each scene. The distribution of the sensors was fixed during the comparison between the algorithms, thus minimizing random factors.

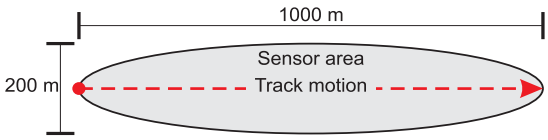


Figure 7 *The scene configuration.*

Since the number of sensors in the different scenes varies, the average overlapping detectable area of the sensors also varies. This value is called the average coverage and is calculated as the fraction between the sum of all detectable areas of all sensors and the area covered by all sensors. The theoretical average coverage for each scene, calculated as the sum of all sensor detectable areas through the theoretical maximum possible covered area, is 1.9 for 20 sensors, 2.9 for 30 sensors and 3.8 for 40 sensors. This value does not tell that much about the actual average coverage, calculated as the sum of all sensor detectable areas through the actual covered area, which is 2.5, 3.7 and 5.1 for 20, 30 respectively 40 sensors.

5.1.3 Results

The outcome of each simulation has been summarized with two values, the average sensor usage and the average target uncertainty area size. The average sensor usage has been adjusted for the sensor activation costs that are caused by the sensor activation algorithm for detection.

In the simulation using only 20 sensors, shown in Figure 8, only the all sensors algorithm stands out among the algorithms. Although the average coverage is 2.5, the all sensors algorithm has an average usage of 3.4 sensors in each cycle. This is explained by the fact that the sensor coverage is higher in the centre of the ellipse and this is where the track is located. The all sensors algorithm can thus be seen as an upper boundary of sensors usage and a lower boundary of the target uncertainty area size.

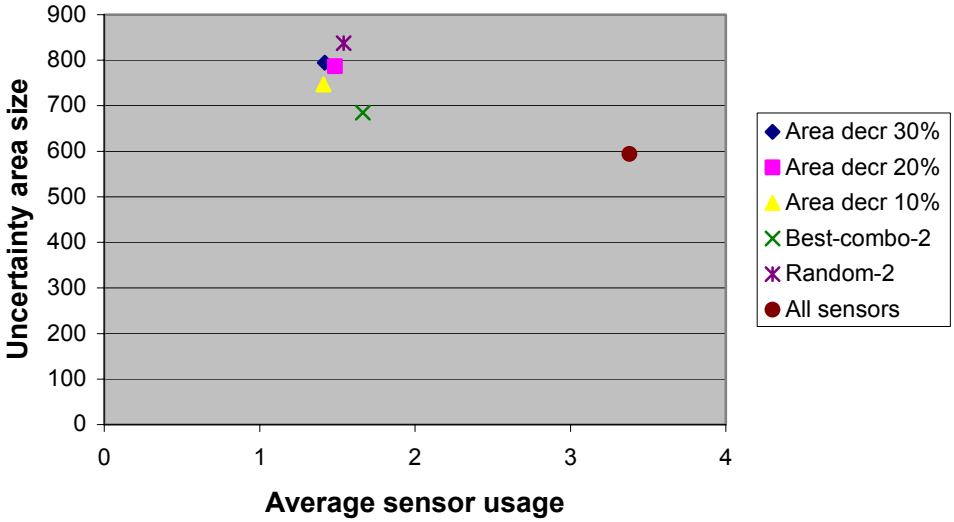


Figure 8 *Comparison between different sensor activation algorithms when running a simulation with 20 sensors.*

The result of the simulation using 30 sensors, with an average coverage of 3.8, is shown in Figure 9. Also here all but one algorithm are clustered together.

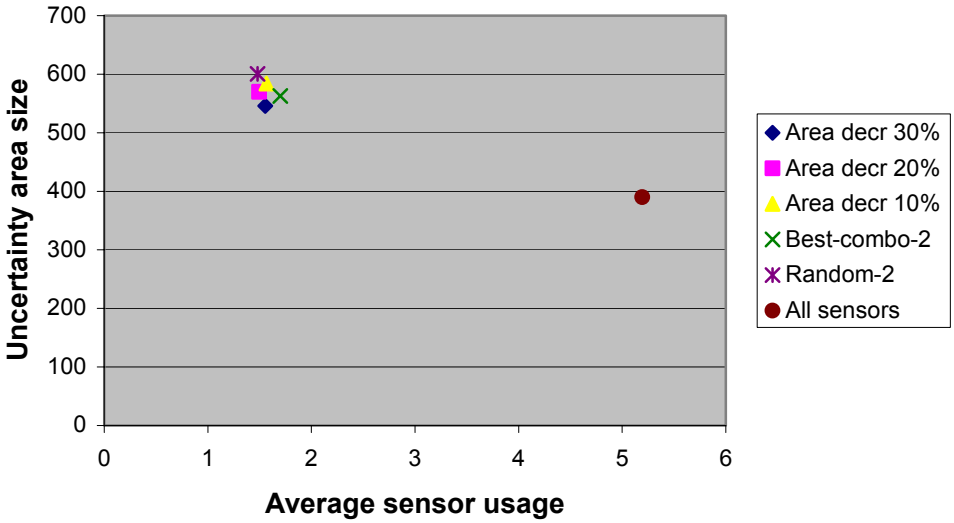


Figure 9 *Comparison between different sensor activation algorithms when running a simulation with 30 sensors.*

The result of the last simulation using 40 sensors, with an average coverage of 5.1, differs slightly from the other simulation results. As shown in Figure 10 the random-2 algorithm has an average uncertainty area that is about 25% larger than the others. The reason why this phenomenon occurs in this simulation and

not in the other two is that now there are more sensors to choose from which results in that the optimal solution is harder to pick at random.

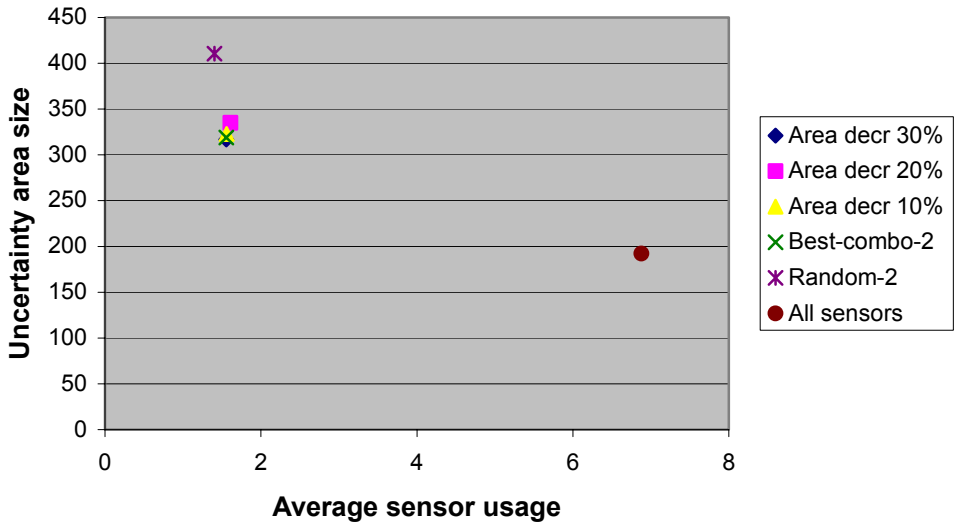


Figure 10 *Comparison between different sensor activation algorithms when running a simulation with 40 sensors.*

5.1.4 Conclusions

The best-combo-2 and the random-2 strategies both choose two sensors whenever they can. The fact that these strategies do not choose the number of sensors to use dynamically makes them less usable when different types of sensors, with different uncertainty area shapes, are used. When the sensor coverage is increased, the random-2 strategy shows a poor result compared to all the other strategies. Since neither the random-2 strategy nor the best-combo-2 strategy show significantly better results than the area decrement strategies, none of these two strategies are chosen as candidates for the sensor activation algorithm for tracking.

The computation power needed for these strategies are in the same order, except for the all sensors and the random-2 strategies that do not calculate the expected uncertainty area of the target.

The average uncertainty area of the decrement strategy is about 30-70% larger than when using the all sensors algorithm. Depending on what the priorities are, track quality or low power consumption, either the area decrement or the all sensors strategy should be used. The use of planning for choosing sensors could reduce the gap in quality between these strategies.

5.2 Multiple Targets

The system was tested with multiple targets. Here a particular test case is presented.

5.2.1 Scene Configuration

A scene containing two crossing targets, see Figure 11, was created and were run in two configurations: One in which the signatures of the targets were identical, and one in which the signatures distinctly differed from each other. Both scenes were 400 cycles long, where the first track (left to right) entered at cycle 0 and left at cycle 399, and the second track (bottom to up) entered at cycle 102 and left at cycle 399.

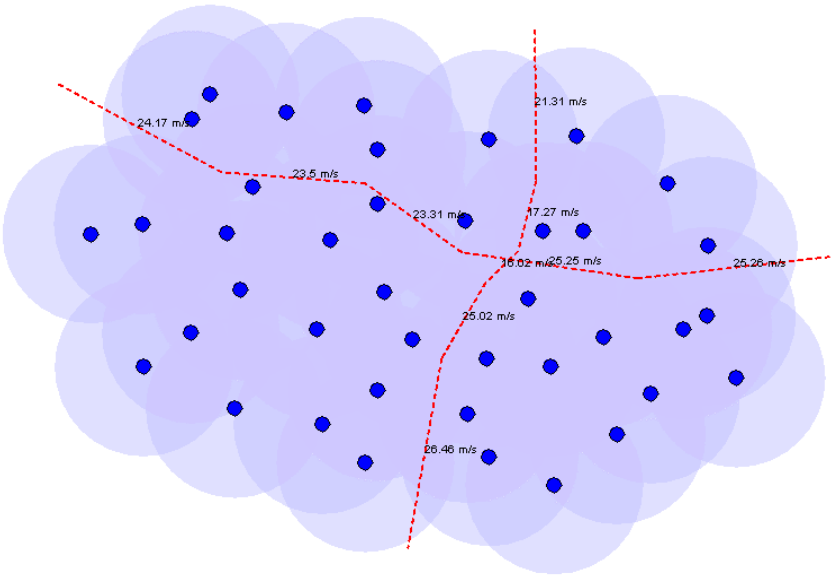


Figure 11 *Scene containing two targets, one from left to right and one from bottom to top.*

5.2.2 Results

In the first test run the two targets had the same acoustic signatures. Since one of the sensor data matching mechanisms is classification of target signatures, this particular scene configuration introduces sensor data conflicts when the targets come too close to each other (see section 3.4.2).

In the second test run the first and the second targets differed very much in the acoustic signature and there was no problem in matching sensor data correctly. Figure 12 shows the result of the different scene configurations.

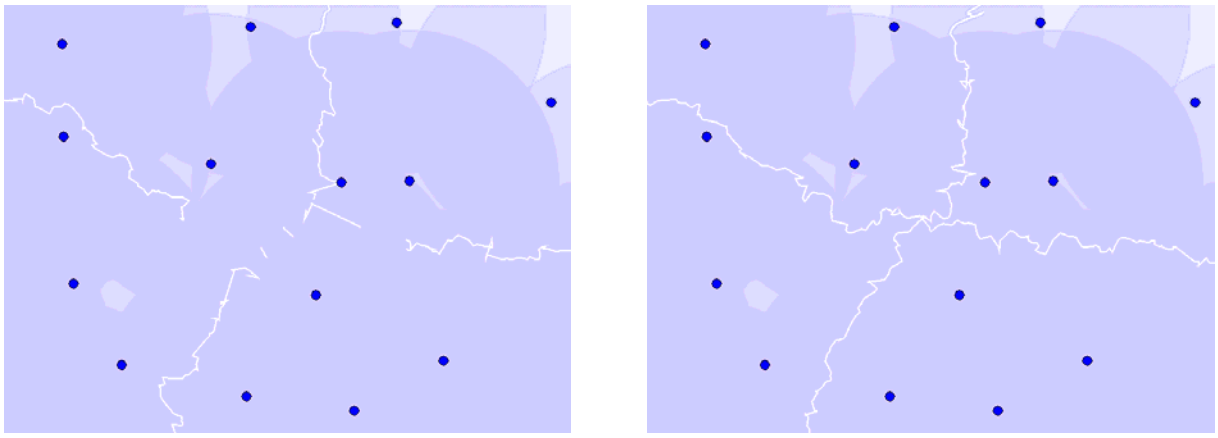


Figure 12 *Simulation result of crossing tracks, with identical signatures (left) and different signatures (right).*

Apart from the crossing point of the targets, both targets were tracked successfully in both scene configurations. Figure 13 shows the difference between the real tracks and the perception of the tracks in the second scene configuration (with different target types).

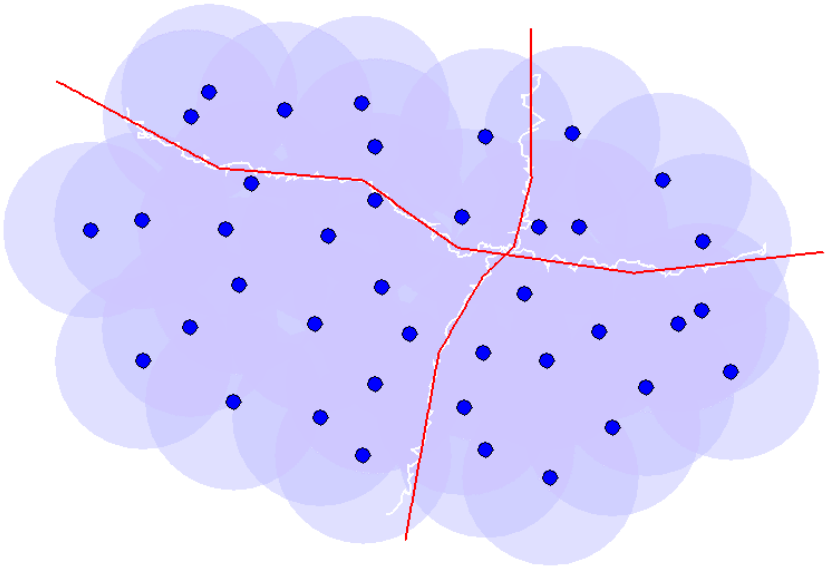


Figure 13 *Simulation result of crossing tracks with different signatures. The real tracks (dark) are overlaid on the perceived tracks (white).*

5.2.3 Conclusions

Tracking of multiple targets is handled in a predicable way. When targets come close to each other, instantaneously decision of which target an observation belongs to might not be possible. Using multiple hypotheses where this decision is postponed could help solving this problem. Tracking of similar targets close to each other is a hard problem that is not always solvable by the sensors at hand.

5.3 Multiple Nodes

An evaluation of how the ratio between the number of nodes and the number of sensors affects the performance has been made. When the system was designed an even distribution of about 5 sensors per node was in mind.

5.3.1 Scene Configuration

Six different scene configurations, where the number of nodes varied, have been used in the evaluation. The scenes consist of 20 sensors, 1 track and 1, 2, 4, 5, 10 respectively 20 nodes. The sensors and the nodes were randomly distributed within an ellipse (400 x 200 m). The track was placed as in the evaluation of the sensor activation algorithm for tracking, se Figure 7 on page 25. The simulations were repeated 10 times for each scene configuration.

5.3.2 Results

The value measured was the number of tracks that was detected. The motivation to measure this value is that when adding more nodes, there is a possibility that sensor data doesn't reach the corresponding track agent. In this case a new track agent is created in the node closest to the source of the sensor data. The previous and the newly created track agent then compete with one another about sensor data, and eventually these two agents will starve each other to death (see sensor data collision in paragraph 3.4.2). The track agents can thus loose track of the target if there are more than one track agent that is tracking the target or if there is a gap in the sensor coverage on the path of the target. The first reason is the most frequent as shown in the figures below.

The average number of tracks in each scene configuration has been calculated and is summarized in Figure 14.

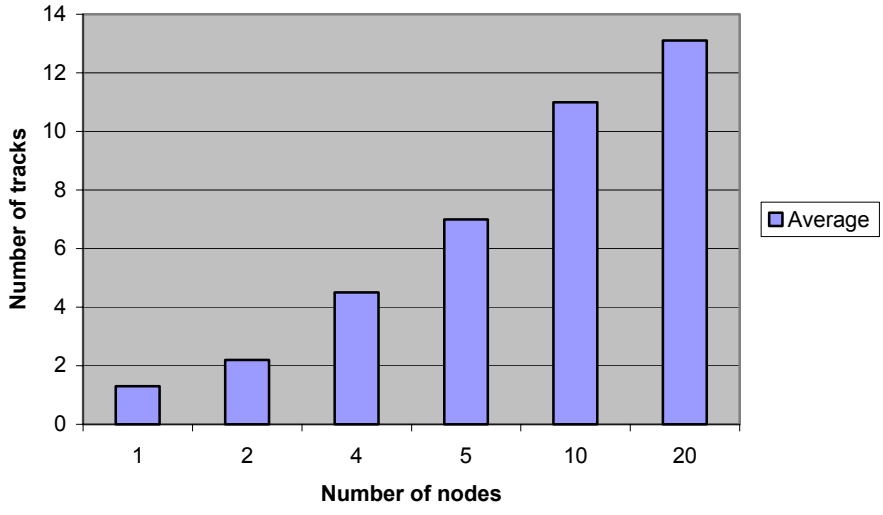


Figure 14 *Average number of tracks against the number of nodes.*

Figure 15 shows the variations in each group. The variations are great, especially in the scene containing 5 nodes. This diagram still shows the same overall result: Adding more nodes will increase the number of tracks (i.e. more frequent loss of tracks).

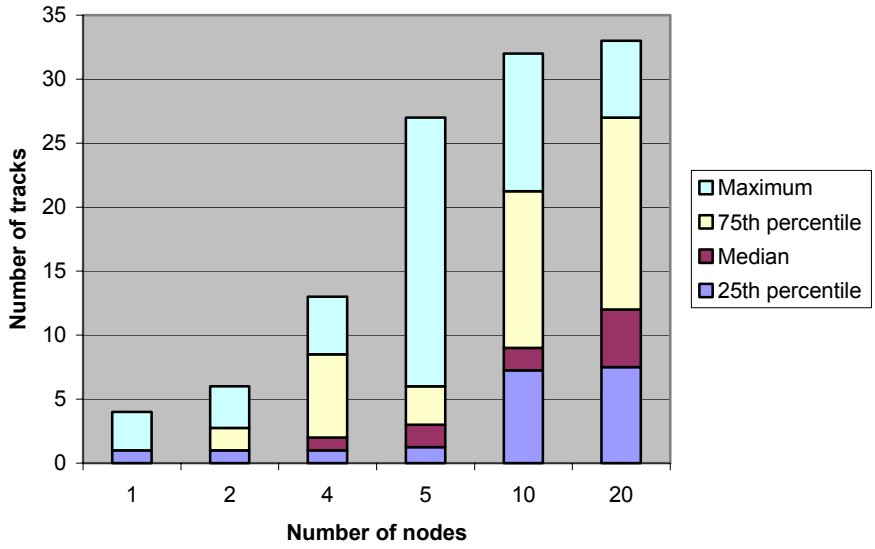


Figure 15 *Number of tracks against the number of nodes.*

5.3.3 Conclusions

The dispatch protocol for sensor data is not satisfactory. The reason is the fact that the dispatch agents don't look in neighbouring nodes for track agents when a local matched track agent has been found. Thus, if two track agents are created at the same cycle in different nodes, they will stop the each other from receiving sensor data from their nodes respectively. The main problem is however: How far should the dispatch agent look for interested track agents? The problem is trivial if the dispatch agent has knowledge of all track agents in the network, but this is not the case.

Chapter 6 Conclusions

6.1 Summary

This report discusses an agent-based architecture for an unattended ground sensor network where agents are used to track, detect and classify targets. This system is designed to be general enough to handle different kinds of sensors. Although only acoustic sensors are simulated in the current stage, other types can be easily added since a common layer of uncertainty areas and signatures is used.

Scalability of the sensor network has been accomplished by removing knowledge of the entire network from the nodes. A node has only knowledge of the neighbouring nodes and its sensors. This way each node has rarely knowledge of more than 16 other nodes, no matter how many nodes there are in the entire network. This means that the network can grow, without increasing requirements on the individual nodes and sensors. Evaluations have however shown that the dispatching protocol is unsatisfactory when adding more nodes.

Sensor activation algorithms for detection and tracking affect the network lifetime to a great extent. Since detection time/probability and track quality is more or less an opposite goal to minimizing sensor power consumption, several tradeoffs have to be made. This subject is discussed further in section 6.2.

The implemented system, which adequately solves the detection and tracking problem (in the case of few nodes), is just in its starting phase. Further development and tests with both real and simulated sensor data are necessary for the system to become a reality.

Applying sensor data to the correct target is a hard problem. Target classification and determination of small target uncertainty areas can significantly help solving this task. Target classification can however not always separate different targets from each other, i.e. when the targets are of (more or less) the same types. When these two techniques fail to match sensor data to the correct target (see section 5.1), other techniques are needed to solve this problem.

6.2 Future Work

There are a vast number of things that might improve this architecture. The following paragraphs each discuss specific areas where improvements are possible.

External Communication

External communication has so far not been concerned. In this work a reference to the "Communication Link" is made available for the agents in all nodes. To facilitate robust external communication more than one node with this capability must exist. There might even be a need for different types of external communication, such as various types of radio communication and satellite communication. Issues as efficient use of energy, transparency of which communication device to use, reliability and robustness should be considered when designing the external communication. From an agent point of view this could lead to new agent types to keep track of the information that should be sent out from the network and how.

Information Dissemination

Information of various types needs to reach many nodes and agents. Examples of this could be various settings, sensor network missions, single commands etc. Often this piece of information or command comes from an external operator via the external communication. This creates the need for some services that can disseminate information to perhaps all nodes, find the mobile agents in the network etc. This should be implemented in an efficient way without unnecessary broadcasts etc., perhaps by keeping track of agents that need to be found for various purposes.

Producer-Consumer Agent Architecture

In the current design, the track agent deals with the task of determining which sensor combination to use. When adding more sensor types this task will grow too big and should therefore be split up into several minor tasks. This could lead to some new specialized agent types that groups sensors together and computes the expected quality of the fused sensor data and the cost in terms of computation, communication and sensor usage. The track agent chooses among the propositions made by these agents. This can be seen as a producer and consumer chain, where track agents buy information from these "fusion agents" that produces information from sensor data. The sensor agents in turn supply the sensor data to the fusion agents.

Node-Sensor Hierarchy

The network could be made less dependent of the nodes if all the sensors had the same capabilities as the nodes. Since the sensors already need computational power for communication, the additional computational power to facilitate

fusion should be easily added at reasonable low cost. The gain is robustness. The current design does however not deal with this kind of flat network topology. The dispatch agents that handle internode communication between sensor agents and track agents only communicate one node away. Their dispatch protocol needs to be revised.

References

[1] Lawrence A. Klein, *Sensor and Data Fusion Concepts and Applications*, Bellingham, WA, SPIE, 1999.

[2] A. Lim, *Architecture for Dynamic Information Dissemination and Fusion in Distributed Sensor Networks*, Proceedings of Fusion 2001 Conference, Montreal, Canada, August 7-10 2001.

[3] J. Agre and L. Clare, An Integrated Architecture for Cooperative Sensing Networks, IEEE Comput. Soc, pp 106-108, May 2000.

[4] Jason Hill et al, *System Architecture Directions for Networked Sensors*, Berkeley, CA, 2000.

[5] S. Kumar and D. Shepherd, *SensIT: Sensor Information Technology For the Warfighter*, Proceedings of Fusion 2001 Conference, Montreal, Canada, August 7-10 2001.

[6] E. Jungert, *Decision Making and Data Fusion in an Interactive Adaptive UGS Network*, Proceedings of the 5th Int. Command and Control Research and Technology Symposium, Canberra, Australia, October 24-26 2000.

[7] M. Wooldridge and N. R. Jennings, *Intelligent Agents: Theory and Practice*, The Engineering Review, 1995.

[8] M. Wooldridge, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, Intelligent Agents,* G. Weiss (Ed.), The MIT Press, London, England, 1999.

[9] N. R. Jennings, *Agent-Oriented Software Engineering*, University of London, 1999.

[10] D. Li, K. Wong, Y. H. Hu and A. Sayeed, *Detection, Classification and Tracking in Distributed Sensor Networks*, Proceedings in Fusion 2001 conference, Montreal, Canada, August 7-10 2001.

[11] T. Kao, D. Mount and A. Saalfeld, *Dynamic Maintenance of Delaunay Triangulations*, Technical Papers 1991 ACSM-ASPRS Annual Convention, pp 219-233, 1991.

[12] L. Guibas and J. Stolfi, *Primitives for the Manipulation of General Subdivisions and Computation of Voronoi Diagrams*, ACM Transactions on Graphics 4, pp 74-123, April 1985.

[13] G. Leach, *Improving Worst-Case Optimal Delaunay Triangulation Algorithms*, Royal Melbourne Institute of Technology, June 1992.

[14] E. Jungert and J. Walter, *Ground sensor nets and intelligent agent system architecture for a network based sensor system*, FOI-R-0317-SE, September 2001.

[15] J. S. Lamancusa, *Engineering Noise Control*, Pennsylvania State University, http://www.me.psu.edu/lamancusa/me458/10_osd.pdf (April 2002), 2000.

Appendix A User's guide

The system consists of three simulation programs: simulation server, communication link and node. There is a tool for creating scene files and viewing simulation files called the scene editor (since it was originally designed for scene editing). There are also simulation tools, a combined graphical user interface for the simulation programs, and a batch simulation tool.

A.1 System Requirements

The system requires jdk1.3.1 or later with the Java API for XML Processing (JAXP) installed. This is what is needed for the system and the tools to compile. If the Jimi SDK is installed, some export features are enabled in the scene editor.

A.2 Scene Editor

The scene editor creates scene XML files and can view simulation XML files, which it can generate and export data from. The scene editor is started by:

```
java se.foi.sensornet.gui.SceneEditor –p <parameters URI>
```

The parameters URI is the location where the parameter file is, for example: `file:M:/project/parameters/parameters.xml`.

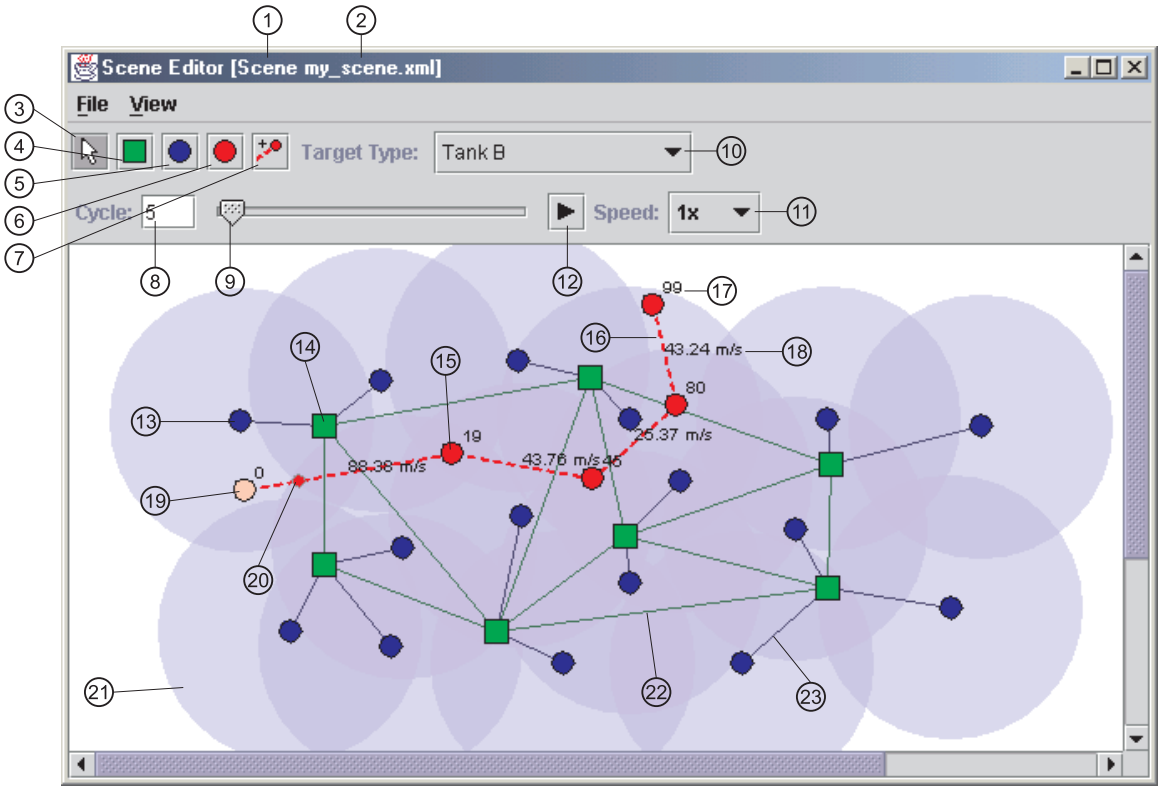


Figure 16 *The scene editor.*

The graphical user interface of the scene editor is shown in Figure 16. The numbers in the figure are described below:

1. Mode – can either be Scene or Simulation.
2. Scene or simulation file name.
3. Selection button.
4. Add node button.
5. Add sensor button.
6. Add target button.
7. Edit track button – only activated when a target is selected.
8. Current cycle field.
9. Current cycle slider.
10. The target or sensor type.
11. The speed to run a scene or simulation.
12. Toggle play/pause scene or simulation.
13. Sensor.
14. Node.

15. Track control point, also referred to as target.
16. Target track.
17. Track control point cycle.
18. Target speed between two track control points.
19. Selected track control point (target).
20. Current target position – depends on current cycle.
21. Sensor detectable area.
22. Node connection.
23. Sensor connection.

The menus in the scene editor graphical user interface are shown in Figure 17.

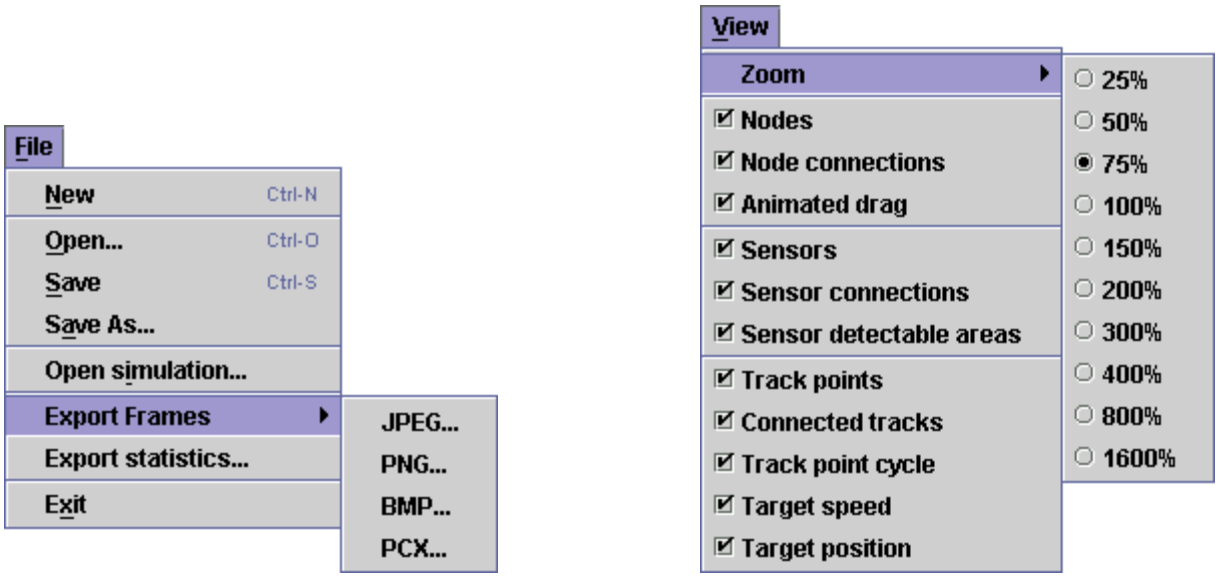


Figure 17 *The file menu (left) and the view menu (right).*

In the *File* menu, the *New*, *Open*, *Save* and *Save As* menu options handle the creation, opening and saving of scene files. *Open simulation* opens a simulation XML file and makes the program go into simulation mode, where all editing tools are disabled. *Export frame*s exports a screen dump of the current scene or simulation for each cycle in the scene or simulation. The screen dump files are named `pic_0001`, `pic_0002` and so on with the corresponding extension. This option is only enabled if the Jimi SDK is installed. *Export statistics* exports statistics of a simulation in a tab separated text file. *Exit* exits the scene editor.

In the *View* menu there are controls that affect the appearance of the drawing area. Setting of the zoom factor and enabling/disabling of different kinds of objects in the drawing area is done here as well.

A.3 Graphical Simulator

The graphical simulator is simply a graphical interface for the simulation server, the communication link, and the node. The simulation server is started by:

```
java se.foi.sensornet.gui.Simulator
                        [-p <default parameter URI>]
                        [-s <default scene URI>]
                        [-o <default output file>]
                        [-h <default host>[:<port>]]
```

The graphical simulator has no required parameters. The specified values are just used as default values in the input fields. When the program starts, the *Simulation Server* tab is selected, see Figure 18.

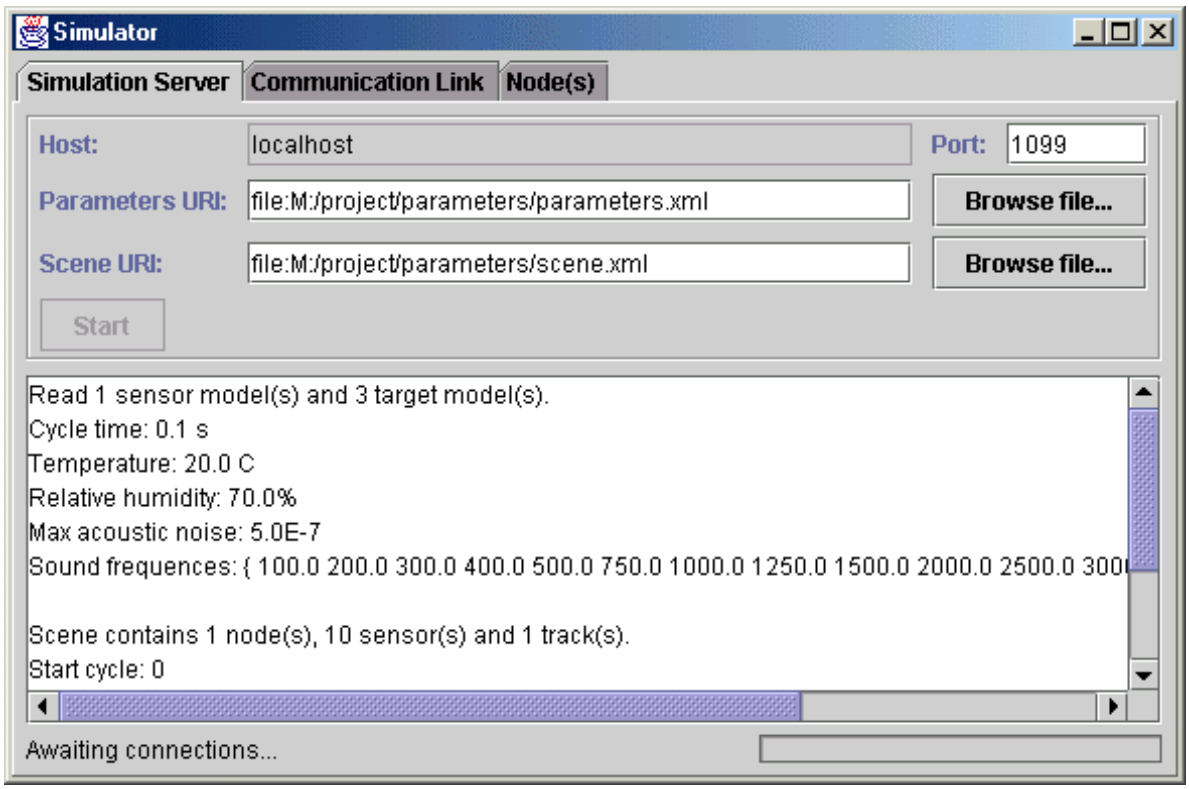


Figure 18 *The Simulation Server tab of the Simulator program.*

Since the simulation server is always started at localhost, the *Host* field contains localhost and is disabled. Only one instance of the simulation server may be run on each host and port, which is specified in the *Port* field. In the *Parameters URI* and the *Scene URI* fields the location of these files are specified. In the text area below the input fields and the *Start* button, the output of the simulation server is written. At the bottom of the window there is a status bar with a status message and a progress bar that shows how far the simulation has proceeded.

When the simulation server has been started, either on this host or another host, the communication link should be started. When the *Communication Link* tab is selected the window should look like Figure 19.

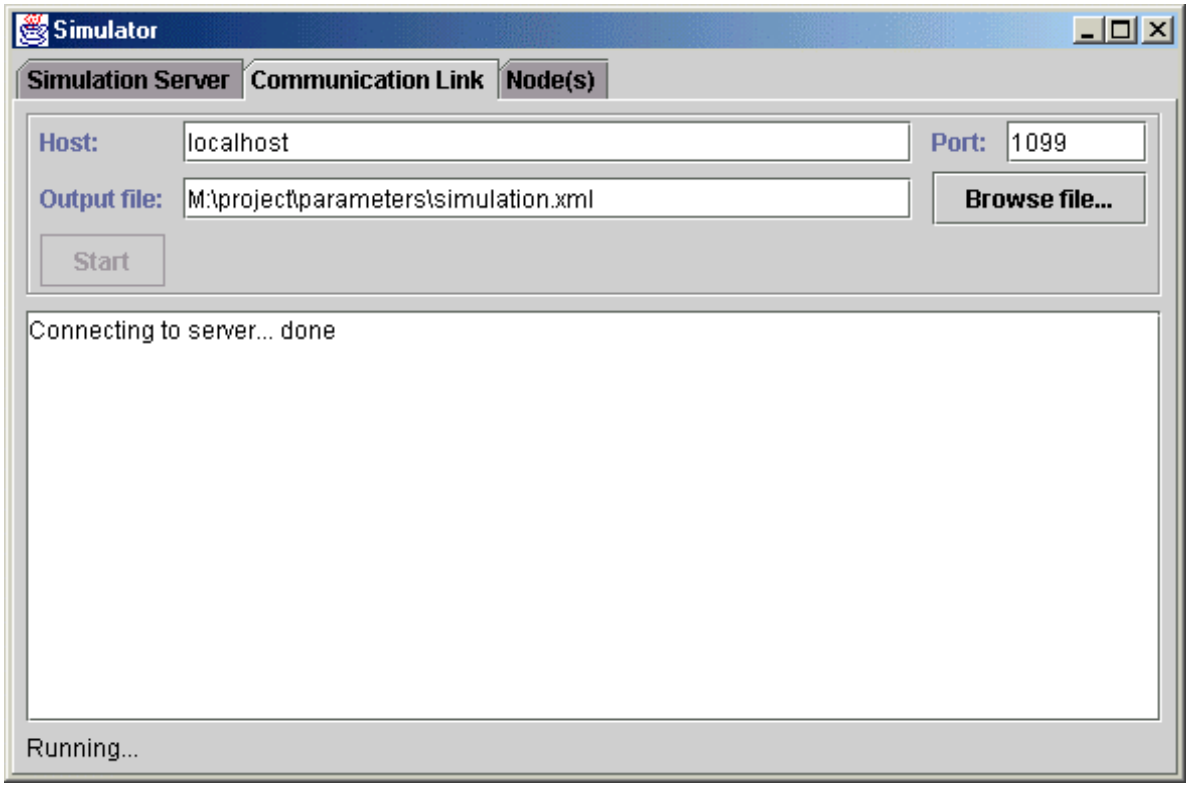


Figure 19 *The Communication Link tab of the Simulator program.*

The host and port of the simulation server are specified in the *Host* and *Port* fields, and the location of the output file is specified in the *Output file* field or browsed with the *Browse file* button. Also here, there is a text area for output and a status bar.

When the simulation server has been started and the communication link is connected, a number of nodes need to be started and connected. The number of nodes to be started depends on the scene file, i.e. how many nodes it contains. The appearance of the *Node(s)* tab is shown in Figure 20.

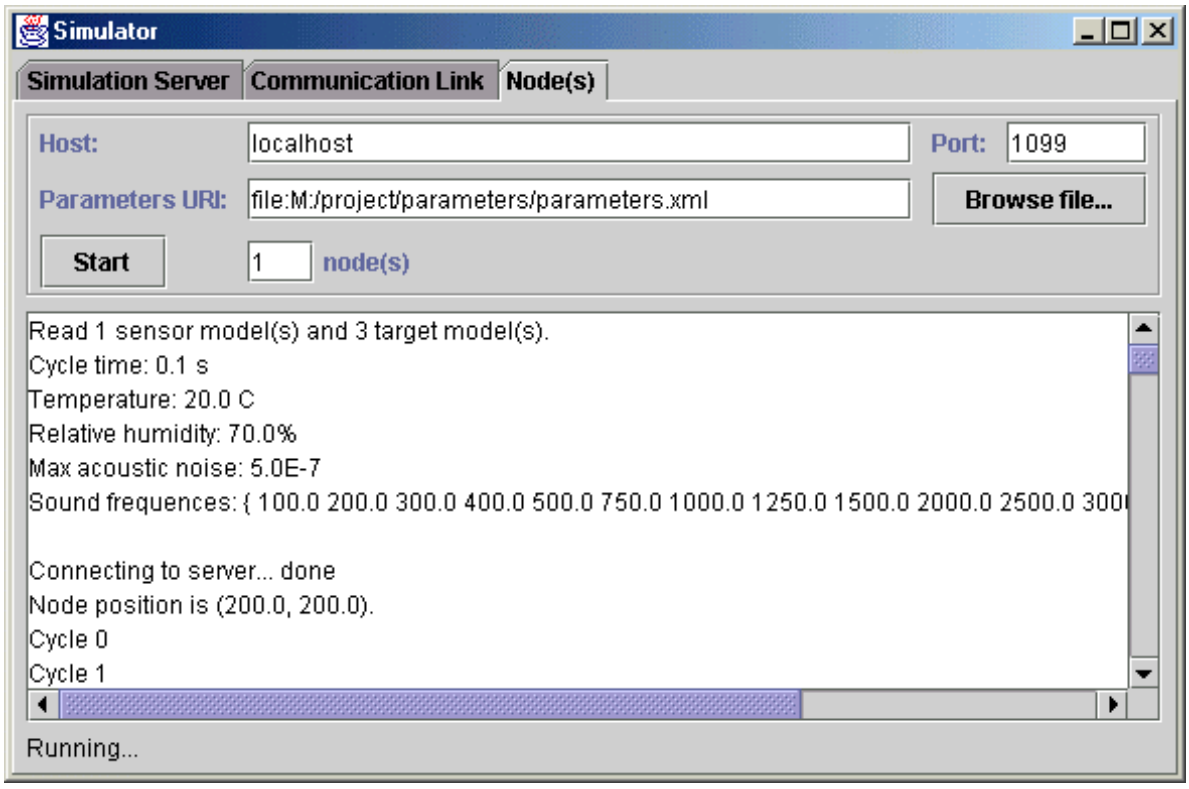


Figure 20 *The Node(s) tab of the Simulator program.*

The host and port of the simulation server are specified in the *Host* and *Port* fields, and the parameters URI is specified in the *Parameters URI* field. The number of nodes to be started is specified in the *node(s)* field. The simulation begins when the required number of nodes is started.

A.4 Simulation Server

The non-graphical version of the simulation server is started by:

```
java se.foi.sensornet.SimulationServerImpl
                        -p <parameter URI>
                        -s <scene URI>
```

A.5 Communication Link

The non-graphical version of the communication link is started by:

```
java se.foi.sensornet.ComLinkImpl
                        -o <output file>
                        [-h <host>[:<port>]]
```

A.6 Node

The non-graphical version of the node is started by:

```
java se.foi.sensornet.NodeImpl
                        -p <parameter URI>
                        [-h <host>[:<port>]]
```

A.7 Batch Simulator

The batch simulator is a non-graphical tool that starts one instance of the simulation server, one instance of the communication link and one or more instances of the node. The batch simulator is started by:

```
java se.foi.sensornet.BatchSimulator
                        -p <parameter URI>
                        -s <scene URI>
                        -o <output file>
                        -n <number of nodes>
```

The batch simulator exits as soon the simulation is complete and the output file has been written.