

On-line Adaptive Context Awareness startend vanuit low-level sensoren

door Kristof Van Laerhoven

Eindwerk ingediend met het oog op
het behalen van de
wetenschappelijke graad van

Licentiaat in de Informatica

Vrije Universiteit Brussel



28 Mei 1999

Thesis promotor: Professor B. Manderick

Vrije Universiteit Brussel
Abstract

ON-LINE ADAPTIVE
CONTEXT AWARENESS
STARTING FROM LOW-
LEVEL SENSORS

door Kristof Van Laerhoven

Thesis Promoter: Professor B. Manderick
Department Informatica

De meeste draagbare apparaten van vandaag weten te weinig over hun context, wat als gevolg heeft dat ze weinig gebruiksvriendelijk zijn. Een GSM heeft bijvoorbeeld geen idee hoe en wanneer de gebruiker gewaarschuwd moet worden als deze opgebeld wordt, vermits er geen informatie is over de huidige situatie. In het algemeen kan je stellen dat vooral draagbare apparaten autonoom – en dus gebruiksvriendelijker – kunnen worden als ze weten in wat voor situatie ze zich bevinden.

De oplossing is het toevoegen van sensoren om het apparaat meer informatie te geven over zijn context. Het probleem is echter dat mensen een context niet beschrijven als de volledige opsomming van de zintuigen. Meestal worden enkel ongebruikelijke elementen (“Het is koud.”) of een algemene beschrijving gegeven (“Ik ben thuis.”, “Ik ben aan het wandelen.”). Als het systeem moet kunnen communiceren met (of leren van) de gebruiker in verband met een context, zal het op een identieke manier zijn context moeten herkennen.

Deze thesis zal vooral de transformatie beschrijven van een grote hoeveelheid en verscheidenheid van sensoren naar een summiere beschrijving van de context, vooraf ingegeven door de gebruiker. Het resultaat is een hybride systeem dat bestaat uit een neurale netwerk met een symbolische laag daarbovenop. Een hardware bord, voorzien van allerlei kleine sensoren vangt de signalen ervan op en digitaliseert ze. Deze sensoren zenden zo hun waarden regelmatig naar een artificieel neurale netwerk, dat verantwoordelijk is voor de initiële verwerking en clustering. Een symbolische module, geïmplementeerd als een Markov keten, maakt het mogelijk de uitkomst te stabiliseren en de gebruiker meer interactie met het systeem te verschaffen.

On-line Adaptive Context Awareness startend vanuit low-level sensoren

by Kristof Van Laerhoven

A thesis submitted in partial
fulfillment of the requirements for
the degree of

Licentiaat in de Informatica

Free University of Brussels



May 28, 1999

Thesis supervisor: Professor B. Manderick

Free University of Brussels

Abstract

ON-LINE ADAPTIVE CONTEXT
AWARENESS STARTING FROM LOW-
LEVEL SENSORS

by Kristof Van Laerhoven

Thesis supervisor: Professor B. Manderick
Department of Computer Science

Along with the sales-figures and the popularity of wearable and portable devices, the importance of their usability and functionality is increasing as well. Most of these devices need to change their behavior according to the context they are currently in. Inadequate knowledge about the context results in a lack of user-friendliness. Mobile phones for example don't know when and how to disturb their users when a call arrives. The solution would be to add various sensors and thus give the host device more knowledge about its context.

However, the way humans describe contexts is not by giving the complete inventory of their sensations. Often, either unusual elements ("It's cold.") or more abstract properties ("I'm home.", "I'm in a dark room.") are noticed and expressed. If the device needs to function in a transparent way for its user or if the user needs to train the device, it needs to recognize contexts in a similar way.

This thesis will focus on the transformation of a multitude of sensory information to a short context description, supplied by the user, in an adaptive and on-line way. The approach is to use an adaptive hybrid system consisting of a connectionist layer, followed by a symbolic layer. Sensor outputs will be periodically sent to a self-organizing artificial neural network architecture, which is responsible for the initial processing and clustering. A symbolic layer, implemented as a Markov chain, provides a predictive component that enables interaction with the user, ensures an enhanced recognition.

Acknowledgments

I would like to thank the following people for their support and contributions to this thesis:

Dr. Bernard Manderick, for his insightful comments, remarks and suggestions in the creation process of this thesis.

Dr. Walter Van de Velde, for introducing me into the TEA research project and for his advice and guidance throughout the whole period of the thesis.

Kofi Asante Aidoo, for taking the time to read this thesis and helping me in improving it and in eliminating a lot of grammar errors.

Adam T. Lindsay, Kristiaan De Paepe, Ronald Schrooten and the rest of the Starlab NV/SA Research staff for giving useful tips and helping me in preparing the experiments.

Albrecht Schmidt and Charles Chen from TecO at the university of Karlsruhe, Antti Takaluoma and Urpo Tuomela from Nokia Research, and Enrico Stupazzini from Omega at the university of Bologna for supplying me with a lot of information and various perspectives on the TEA project and context awareness.

And finally, my parents for their financial and moral support that made this thesis possible.

Kristof Van Laerhoven

Vrije Universiteit Brussel

May 1999

Contents

CONTEXT AWARENESS	5
1.1. CHARACTERISTICS OF CONTEXT AWARENESS	5
1.2. CONTEXT AWARENESS IN THE PAST	6
1.3. TECHNOLOGY FOR ENABLING AWARENESS	7
1.4. WHY CONTEXT AWARENESS	9
1.5. STRUCTURE OF THIS THESIS	11
FROM SENSORS TO CONTEXTS	13
2.1. THE OBJECTIVE	13
2.2. ACQUIRING DATA FROM SENSORS	14
2.3. EXTRACTING CUES	16
2.4. A FIRST ANALYSIS OF THE DATA	17
2.5. ARCHITECTURE OF THE TEA-SYSTEM	18
CLUSTERING THE SENSOR DATA	20
3.1. CLUSTERING ALGORITHMS IN GENERAL	20
3.2. SENSOR WEIGHTING AND OTHER PRE-PROCESSING TECHNIQUES	23
3.3. NEURAL NETWORKS	24
3.4. MAPPING WITH NEURAL NETWORKS	26
3.4.1. <i>The Kohonen Self-Organizing Map</i>	26
3.4.2. <i>Curse of Dimensionality</i>	31
3.4.3. <i>Stability versus Plasticity</i>	31
3.4.4. <i>Temporal variations on Kohonen's SOM</i>	32
3.4.5. <i>Tracking the activation</i>	33
3.4.6. <i>The Adaptive Resonance Theory</i>	35
3.4.7. <i>Other unsupervised neural networks</i>	37
3.4. EXPERIMENTS AND RESULTS	38
THE INTERMEDIATE LAYER: DIVIDING THE INPUT SPACE	48
4.1. DEALING WITH HIGH DIMENSIONALITY	48
4.2. EXPERIMENTS AND RESULTS	50
CLASSIFICATION ALGORITHMS	54
5.1. SIMPLE ASSOCIATION	54
5.2. DISTANCE WEIGHTED K NEAREST NEIGHBOURS	54
5.3. RADIAL BASIS FUNCTION NETWORKS	56
5.4. KOHONEN'S LVQ	57
5.4.1. <i>LVQ1</i>	57
5.4.2. <i>LVQ2</i>	58
5.4.3. <i>LVQ3</i>	59
5.5. MINIMIZING THE USER INTERVENTION	60
THE SUPERVISING LAYER	62
6.1. A PROBABILISTIC FINITE STATE MACHINE	62
6.2. MARKOV CHAINS	63
6.3. INTRODUCING ADAPTIVITY	64
6.4. CLUSTERS	66
6.5. EXPERIMENTS AND RESULTS	66
IMPROVING THE PERFORMANCE	68

7.1. THE ARCHITECTURE.....	68
7.2. PARAMETERS OF THE SOM AND RSOM.....	69
7.3. WHAT (COMBINATION OF) SENSORS WILL PERFORM BEST?	69
FUTURE WORK.....	71
8.1. IMPLEMENTATION OF USER SUPERVISION AND INTERACTION.....	71
8.1.1. <i>No user feedback</i>	71
8.1.2. <i>Some user feedback</i>	72
8.2. NON-EXCLUSIVE CONTEXTS	72
8.3. RULES EXTRACTION	73
8.4. TOP-DOWN STABILISATION MECHANISM.....	74
CONCLUSIONS	76
9.1. THE ARCHITECTURE.....	76
9.2. ACCOMPLISHMENTS	78
BIBLIOGRAPHY	79

List of Figures

Figure 1: The 3D context model (source: [27])	6
Figure 2: Approach of TEA (and this thesis) to reach context awareness.....	8
Figure 3: Context Awareness and Personal augmentation (source: Nitin Sahwney, MIT Media Lab).....	9
Figure 4: The situation of Context Awareness in the application structure.....	10
Figure 5: The parts of the architecture discussed in chapters 3, 4, 5 and 6.	11
Figure 6: The acquisition process.	15
Figure 7: Example of a time series plot of sensor outputs in five different contexts.	17
Figure 8: Phase space plot of the data coming from three contexts.....	18
Figure 9: Layered architecture of the TEA project.....	19
Figure 10: a) points in \mathfrak{R}^2 , b) corresponding Voronoi tessellation and c) corresponding Delaunay triangulation. (source: [11])	21
Figure 11: Structure of an artificial neuron.....	25
Figure 12: Structure of a Kohonen SOM.....	27
Figure 13: Example of the output layer of a SOM.....	29
Figure 14 a: The SOM output layer. b: The corresponding activation surface.....	33
Figure 15: Tracking the input while walking from the inside to the outside context.	35
Figure 16a: The output layer of the KSOM after one epoch.	40
Figure 16b: The output layer of the KSOM after two epochs.	40
Figure 16c: The output layer of the KSOM after three epochs.....	41
Figure 16d: The output layer of the KSOM after four epochs.	41
Figure 16e: The output layer of the KSOM after 5 epochs.....	42
Figure 17a: Tracking the input on a frozen activation surface during the first context.....	43
Figure 17b: Tracking the input on a frozen activation surface during a second context.....	44
Figure 17c: Tracking the input on a frozen activation surface during a third context.	45
Figure 17d: Tracking the input on a frozen activation surface during the fourth context.....	46
Figure 17e: Finished tracking the inputs on an activation surface during five contexts.	47
Figure 18: An example of the architecture of the system so far.	49
Figure 19 : Recognition of contexts with two SOMs in parallel	52
Figure 21: The window of the LVQ2 algorithm.....	59
Figure 22: Schematics of the user involvement.	61
Figure 23: Example diagram of the Markov model.....	63
Figure 24: Example diagram of the Markov model of clusters.	65
Figure 25: Example diagram of a hierarchical architecture.....	73
Figure 26: Example diagram of the final architecture.	77

List of tables

Table 1: The leader clustering algorithm.	23
Table 2: The algorithm to produce Kohonen SOMs.	30
Table 3: Facts and Figures of the experiment.	50
Table 4: The distance-weighted k nearest neighbour algorithm.	55
Table 5: The LVQ1 algorithm.	58
Table 6: The LVQ3 algorithm.	60
Table 7: The steps to update the probabilities in the Markov chain.	65

Chapter 1

Context Awareness

Making machines aware of their contexts is not a new concept. A lot of applications already use sensors to get an idea of what is happening in the surrounding environment. However, the number and variety of sensors is usually minimal and the recognition process is kept very simple and compact. Establishing a high-level notion of context, based on the output of a group of simple sensors is not very common. The field of robotics [23] started very early with this approach (subsumption architecture for autonomous robots) and has probably been responsible for most of the progress that has been made so far.

This chapter will give an introduction to the concept of context awareness followed by a short overview of related context aware projects. An explanation will also be given on how this thesis is situated in the framework of the European Commission Esprit project, TEA. In conclusion, the structure of the proposed architecture is given, along with the further composition of this thesis.

1.1. Characteristics of context awareness

The notion of context awareness for devices itself can be split up into three components (as seen in Figure 1): activity, environment and self. The activity describes the task the user is performing at the moment, or more generally what his or her behaviour is. This aspect of context is focused on the user of the device, and his or her habits. The environment describes what the status is of the physical and social surroundings of the user. The current location, the activities in the environment and other external properties like for instance temperature or humidity belong to this axis. Finally, the self component contains the status of the device itself. This third point of view on context awareness has not been researched as much as the other two, but is a very interesting one in the scope of cognitive sciences.

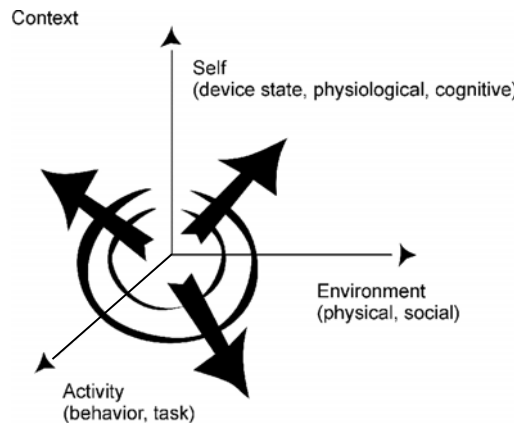


Figure 1: The 3D context model (source: [27])

1.2. Context Awareness in the past

In the past, a lot of context awareness research was often restricted to location awareness, thus providing the device knowledge about location only. The hardware was usually based on either beacons or global positioning systems (GPS) that are able to flawlessly recognise the current location or coordinates, but fail to say anything more about the context. Such an approach can be very expensive and it would be a great deal cheaper to use a lot of different low-level sensors. The combination of these sensors leads to a detailed description of the surroundings, which gives more than just an indication of the location. This alternative is less accurate, but it is superior if the application does not expect the system to be fully aware of its position and a simple hunch of location is sufficient. Context awareness is more than a mere recognition of a geographical location [26]; it is about recognising the current situation.

Many applications exist that use location awareness to enhance the services provided by a device. Most applications use the knowledge about the location as a sort of guide through buildings, museums, cities and so on [1]. An uncommon application is proposed by Eriksson and Finne [8], which uses music or soundscapes (sounds, or harmony - landscapes) to “inject” feelings into the user based on location awareness via GPS. More concrete, if the user walks into a ‘bad’ neighbourhood, he or she hears sounds that give the appropriate feelings.

Other projects focussed on specific sensors like microphones, photodiodes (light sensors) or movement sensors to derive a (usually basic) notion of context. One particular project by Sawhney at MIT [25] focused on solely the output of a microphone to recognise different kinds of noise that might be specific to a location. A DAT recorder was used to record both a training set and a test set of several hours in different environments. A recurrent neural network and a nearest neighbour algorithm were implemented to recognise various “situational contexts” like speech, subway, traffic and so on. The goal of the experiment was not to do adaptive real-time context recognition, but focussed more on the feasibility of context recognition based solely on sound. For some contexts like speech and subway, recognition statistics seem very good.

Perhaps the best known device for supporting context-aware computing is the Smart Badge system of Olivetti (described in [3]). The Badge is equipped with a standard IR transceiver and a collection of sensors, which allow the Badge to sense and transmit the user's environmental information and user's ID back to the network. In the network, a Badge server is used to analyze the sensor data and provide the environment information to proper users. The Smart Badge also provides enhanced I/O capability, which makes it possible to connect displays, audio, and video devices. One of the best-known applications of the smart badge is tracking people inside a building, with the help of ‘smart doors’ that register which people enter the rooms.

1.3. Technology for Enabling Awareness

The approach of this thesis on achieving context awareness fits in the framework of TEA [9] (Technology for Enabling Awareness), a project funded by the European Commission Esprit Program. Researchers from four laboratories in four different countries joined forces for this project: Starlab NV/SA in Belgium, TecO (University of Karlsruhe) in Germany, Omega Generation (University of Bologna) in Italy and Nokia Research in Finland. TEA's objective is to research context awareness in the field of handheld and wearable devices such as mobile phones, personal digital assistants or even laptops. Low-cost and widely available sensors on physical parameters and information are the focus in this project. Starting with this information, the TEA system should be able to determine in what situation the user is at every moment. The aim is, in the words of the TEA document: “To produce an affordable add-on component to existing portable communication and

computation devices that adds awareness of location and activity, and thus enables context sensitive device control.”

The accomplishments that have been given birth after a first year are promising. A prototype hardware board has been created to control the sensor outputs in a flexible way. Sensors can easily be added or removed and the processing to be performed on the collected data can be done via a PC connected to it with a serial cable. The presence of the serial connection, combined with the small size of the hardware board, makes that the whole acquisition hardware remains portable. Furthermore, experiments were prepared and executed that merged this hardware module with existing devices such as a mobile phone and a personal digital assistant (PDA). These experiments, demonstrating the possibilities of context awareness in the field of mobile communications, were both promising and encouraging towards the future of the project.

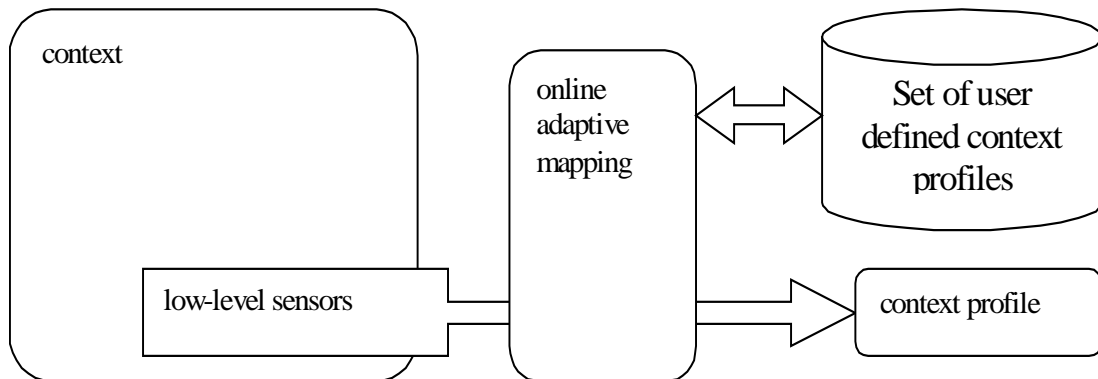


Figure 2: Approach of TEA (and this thesis) to reach context awareness.

The fact that TEA starts with the raw data of low-level sensors is not the only unusual property of the project. TEA also envisions the use of adaptive algorithms to reach adaptive context awareness, which makes it possible to learn and adapt to specific contexts for a specific user. This results in a context aware system that will adjust itself towards the user and the contexts he or she will frequently visit, thus improving the quality and speed of recognition of these contexts. Another requirement is that human supervision must be kept to a minimum, since the system should be designed to bother its user as little as possible and thus to obtain a maximum degree of user-friendliness. Consequently, augmenting the autonomy of the device increases its user-friendliness.

A context aware device can be mapped onto a two-dimensional plane that describes the amount of involvement required from the user and the level of situational awareness. Sunglasses are an example of not very intelligent, passive devices. They are not very intelligent because they have no clue whether the sun is shining or not. They are very passive because they operate usually in an autonomous way, requiring very little user-involvement – the user only has to decide whether to put them on or off. More intelligent sunglasses exist that adapt themselves to the brightness of the sunlight; these would be placed higher on the situational awareness scale, but they remain as passive as their less-intelligent versions. The objective of TEA is to make an adaptive system for a device to make it as aware as possible and at the same time as passive as possible towards its user.

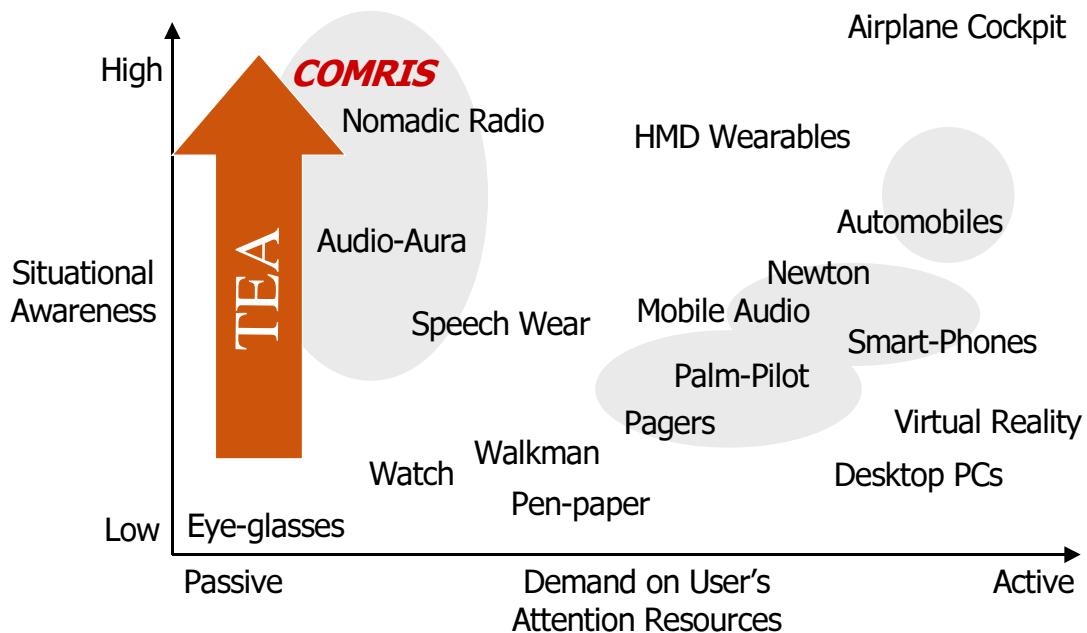


Figure 3: Context Awareness and Personal augmentation (source: Nitin Sahwney, MIT Media Lab)

1.4. Why context awareness

Appliances are becoming increasingly ubiquitous, increases in the use of portable computers and cellular telephones are clearly visible and the image of having a portable device is not anymore reserved to a restricted group of people. Research and development in information technology is moving away from desktop based computing towards more portable and task specific applications. Most of these

portable devices should behave differently depending on what situation they and their users are in.

Along with the sales-figures and the popularity of these devices, the importance of the usability and functionality is increasing as well. Appliances that know more about their environment will be able to function better and will give their users a better service. A device that knows about its own environment and that of its user could transparently adapt to the situation, leading to the idea of the invisible computer as discussed by Weiser [30]. This in turn is a step towards a disappearing interface as demanded by Norman [20]. Improving the interaction with such a device is not possible without augmenting its context awareness. The appliance will be capable of giving better defaults for the situation and could automatically make choices that the user normally would have to make.

The application model is thus changed in structure as illustrated in Figure 4. The classic structure has a model of the user and a model of the context, derived from solely the input of the application. In contrast, the context aware application contains a context model that is now altered by the context aware system, providing the application new and possibly convenient information. Consequently, the context model has a higher value, resulting in a better application.

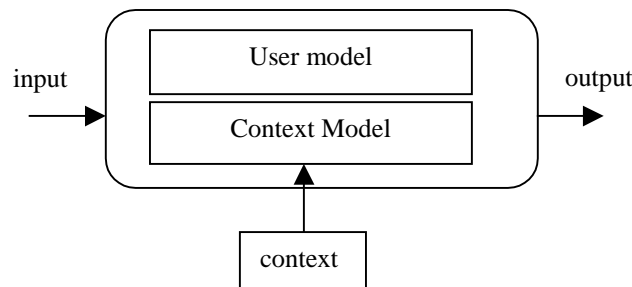


Figure 4: The situation of Context Awareness in the application structure

Apart from the enhanced user-device interface, context awareness can also improve the quality of services that already exist, like providing context-related information, behaviour and communication.

Another reason to implement (more) context awareness is the ability to add sensors that can enable the device to perform monitoring and extended sensing. A simple GSM could be turned into an appliance that observes one's health if some biosensors were added. The apparatus would not have to stick to mere monitoring,

but could also take appropriate actions if the user's health would be endangered. Other possibilities are security- and safety- related applications.

1.5. Structure of this thesis

Chapter 2 describes the framework where the on-line adaptive context learning architecture is located in. Chapters 3, 4, 5 and 6 describe the layers or modules of the hybrid system (as illustrated in Figure 5) that is proposed in this thesis to transform raw sensor data to context profiles.

Chapter 3 deals with various approaches and algorithms for clustering the sensor data. Chapter 4 presents a hierarchy of clustering algorithms that is able to solve some of the problems that emerged in Chapter 3. Chapter 5 introduces a classification layer that labels the clusters from the previous layer with short context descriptions (context profiles). Chapter 6 describes the final layer, which provides an element of supervision to the system.

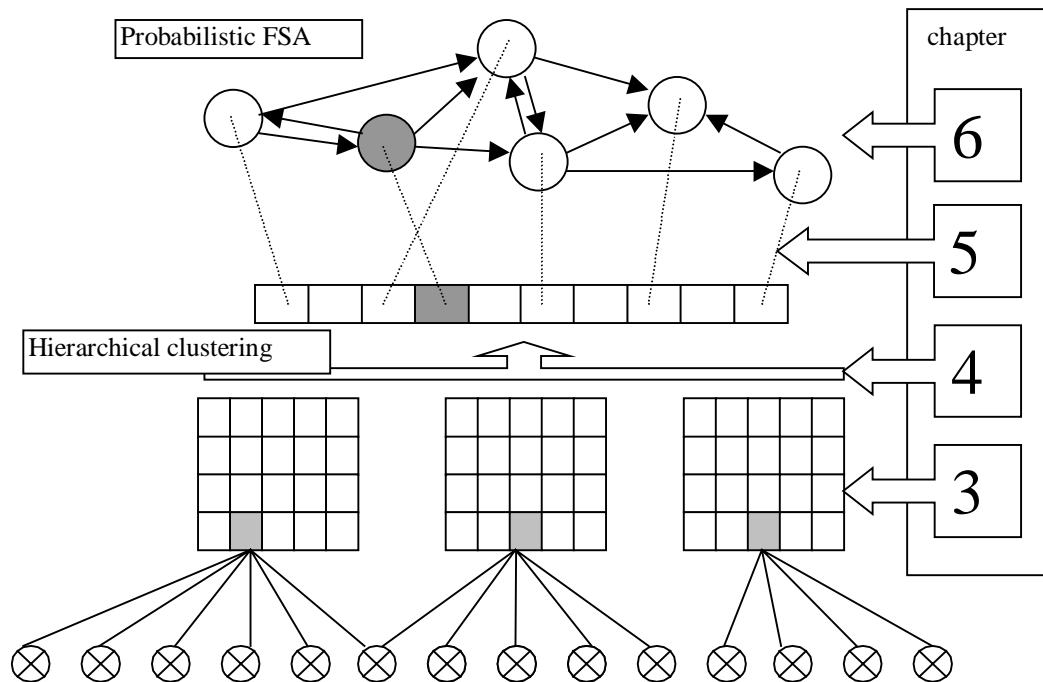


Figure 5: The parts of the architecture discussed in chapters 3, 4, 5 and 6.

Chapter 7 evaluates some possible improvements and gives an outline of performance issues for the system. Finally, chapters 8 and 9 will conclude with future work and the conclusions.

Chapter 2

From Sensors to Contexts

This chapter will focus on both the basis and the goal of the system. A concise explanation of the concepts of sensors, cues and contexts will be followed by more practical information on how the experiments were conducted and initial analysis results. An elaboration on how the adaptive system fits into the bigger framework of the TEA project will conclude this chapter.

2.1. The objective

The source of the system's input is the set S of basic sensors. In theory, a sensor can be just about anything: a light sensor, a microphone, a global positioning system (GPS), a camera or even a human. In literature, this last kind of sensor is usually referred to as a logical sensor, while the other examples are called physical sensors. The only requirement a sensor should comply with is that it should periodically produce a value that resembles a measurement of a physical phenomenon in its real world context. The formation of such a signal is a digital and unstructured sub-symbolic value. A context description resides usually on a higher level than the outputs of the sensors. This section will provide a general overview on how low-level sensor data can be transformed into a high-level context description.

At every moment in time, the sensors provide the system with a set of values that give a description of the context at that time. However, this description is very difficult to interpret – and even unreadable as the number of sensors increases - for humans. This is not a problem if the application does not demand user-involvement or user supervision or if it is merely used for controlling purposes. If the user has to be kept informed on the context-profiles, however, it is necessary to build a bridge between the human and machine context description. The solution used in this thesis is to transform the sensor readings to a context profile, for instance “*in a busy office*” in stead of (light=90%, microphone=77%, ..., infrared=89%) and “*in a meeting*” in stead of (light=90%, microphone=15%, ..., infrared=8%).

The output of all sensors should thus lead to a context profile that was pre-defined by the user. This profile is usually a crude generalisation that leaves out a lot

of other information, so that the user can easily and quickly define a certain context and recognise it later on. Profiles like “running”, for example, will probably only need the data from infrared and acceleration sensors – the contribution of all other sensors would be minimal. Another property of the context profiles is that they have to reflect the application they are used for. It is in general very common to have contexts that might overlap, like for example “running” and “at the park”. If the application should demand that one context must be chosen, it will depend on the nature of this application which context will finally be elected as the winner.

The use and recognition of exclusive or non-exclusive context profiles depends only on the application the system is intended for. The architecture in this thesis is able to implement both, but is restricted to non-overlapping context profiles. Overlapping contexts and hierarchies of contexts are treated in the future discussions chapter (Chapter 8).

To recapitulate, the objectives of this system are:

1. The system has to transform raw data from several low-level sensors to a high-level, short context description.
2. The system has to be adaptive, to enable the learning of new contexts and to augment the recognition of learned contexts.
3. The learning must be done on-line (during execution) and as real-time as possible.
4. Since the user has to teach the system what he or she means with a specific context, the system must fully exploit this training-phase. This will result in less user-intervention and, consequently, more user-friendliness.

2.2. Acquiring data from sensors

A small, portable hardware device called “TEA Sensor-Board” was responsible for acquiring the data that was used in this thesis. It was developed within the TEA project, to provide a prototype that enables to acquire raw sensor data (see chapter one for a more complete description). The sensor board is equipped with many different low-level sensors and a chip that digitises the analog signals from the sensors at a certain rate. This frequency can be different for every sensor and depends on how quickly a value of a certain sensor can change. An acceleration sensor for instance is probed about 100 times per second, while a temperature sensor is probed only once a second. This means it could be theoretically possible to create

a set of input values from the sensors every millisecond, although this probably would be too frequent for the system.

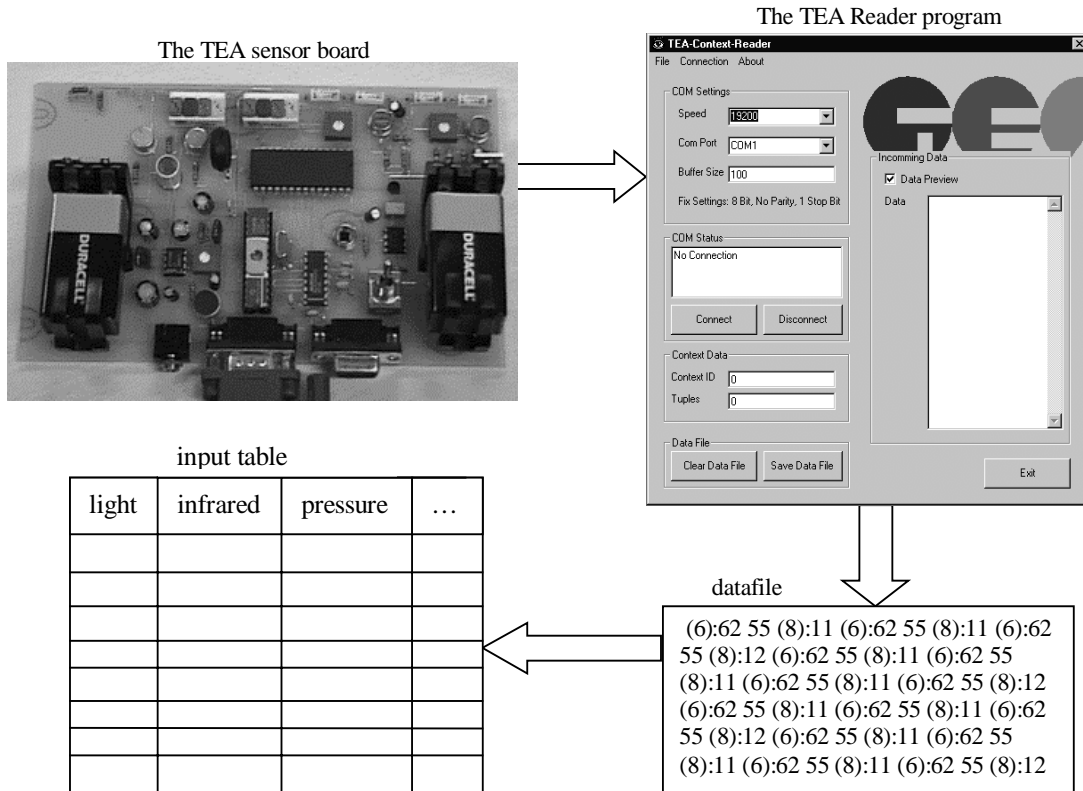


Figure 6: The acquisition process.

It is not necessary to update the input of the system that often, and in practical applications a second will be enough. Beside the raw sensor values, it is also possible to use the mean, the variance or another value that describes the output over that second for sensors that are highly time varying such as the accelerometers, infrared sensors or microphones. This will partially make up for any critical values that could be thrown away by enlarging the update period. This extraction of information from raw sensor data will be discussed in section 2.3. (Extracting Cues). Used sensors include light sensors, accelerometers, infrared sensors, microphones, CO gas sensors, temperature sensors and atmospheric pressure sensors.

The TEA board was connected to a fast laptop computer during the experiments, rendering the whole system mobile to carry it around from context to

context. On the laptop, monitor software (TEAreader, programmed by one of the TEA partners¹) was installed to read the raw data and write it out to a data file on the harddisk for later analysis. Before a neural network clustering layer can process the data, the sensor values have to be copied into an input table that groups the numbers by timesteps. This input table will also contain cues from the frequently probed sensors, which will be discussed later in the next section. Figure 6 shows the major steps in the acquisition process.

2.3. Extracting cues

Every sensor that is provided with the necessary power will spit out a large amount of values over time. Some of them will produce such a large stream of data that it is almost impossible to use this directly as input for the final recognition system. The values of the light sensor can be replaced (as described earlier) by the mean and the variance. Other transformations and filters can also be applied for this purpose. Another – and perhaps better - example is the microphone since it produces an even larger amount of values. Chen [7] has researched the output of this particular sensor and noticed a few values that can be calculated in real-time and that are able to give a very clear idea of the current context. Filters and transformations of the raw sensor data are also examples of possible cues.

These values are called *cues* (sometimes the term features is also used in this context), because they indicate what kind of data the sensor is producing without actually giving directly the output. The cues can be interpreted as values that are the results of often small and quick calculations on data that is sent very frequently. This method does not just enable the learning system to be as near real-time as possible, it also optimises its performance since the cues usually give a better interpretation of the data.

Although the values from not all the sensors are being described by cues, it is possible to state that the recognition system starts with cues instead of sensor data. Even if the values from a sensor are not pre-processed to a cue, it could be called a cue if other sensors were pre-processed. The mere fact that the choice was made to let the value be a cue of itself justifies this.

¹ TecO, the University of Karlsruhe Telecommunication Office

2.4. A first analysis of the data

A first visualisation of the data is established by just plotting the time series of all sensors in the same plot. This graph is essential in evaluating the acquisition process in two ways. First, it will usually give a pretty good clue whether the system is able to distinguish contexts from each other or not, since the different contexts show different patterns in the sensor readings. Secondly, it is a good method to visualise and evaluate the performance of a particular sensor. The plot in Figure 7 shows an example of 8 sensor outputs during 832 seconds, taken in five basic contexts: “inside a dark office”, “inside an artificially lit office”, “moving inside an artificially lit office”, “outside still” and “outside moving”. The TEA sensorboard was turned off after each context was ‘recorded’ and turned back on when it was brought in a new context. This is mainly the reason why the transitions of contexts are very short or even invisible. The time series plot is also a good tool to predict which contexts will be hard and which contexts will be effortless in learning: the last two contexts will probably demand more resources (time, memory and/or processing) than the others. In this graph is also the importance illustrated of temporal pattern recognition: some of the sensors in the first contexts are harmonic signals that could be recognised by specific algorithms that are time-sensitive.

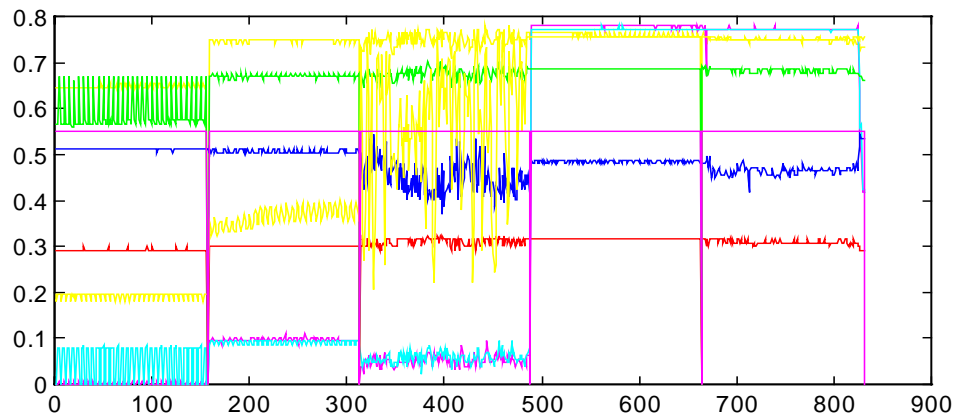


Figure 7: Example of a time series plot of sensor outputs in five different contexts.

When a sensor produces a value, this value could be considered as a component of a vector, which has a dimensionality equal to the number of available sensors. This leads to a second method of visualisation, in which each point in an Euclidean space could represent this vector. Plotting this space requires that the

dimension of the space is less than - or equal to three. Thus, some sensors have to be ignored if there are more than three sensors available. The next plot (Figure 8) shows a three dimensional phase space, in which the input vectors from the first three contexts of the picture above are traced. The last two contexts were not traced, because their vectors were placed very far away, and there was no observable distinction between them. The reason for this correlation can be deduced from the time series plot: the two contexts have too many values in common to be separable in this clustering visualisation.

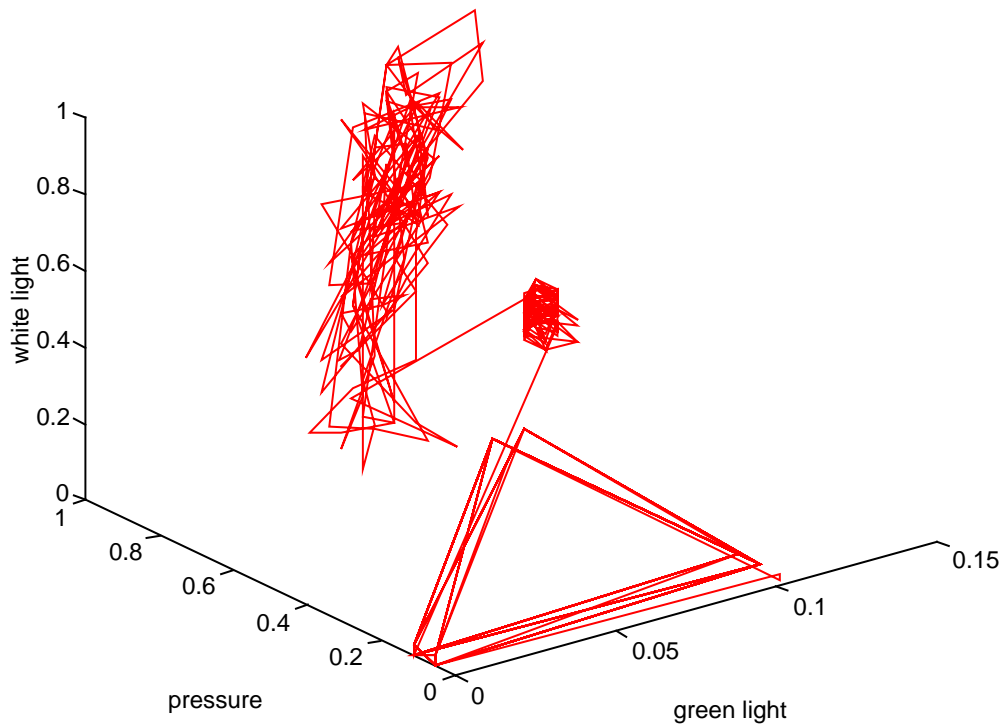


Figure 8: Phase space plot of the data coming from three contexts.

2.5. Architecture of the TEA-system

The TEA system consists of the four layers that are depicted in Figure 9. The first layer is the sensor layer, which represents the acquisition of the raw data from the low-level sensors. In the second layer, the output of every sensor is described by one or more cues, which are usually more abstract than the sensor data itself. The third layer is probably the most difficult one: mapping the cues to a context profile

that was given by the user, since it involves user-interaction. Finally, the fourth layer uses the context information to change the behaviour of the application or device according to the current context. The result is four-layer system that is “context aware”. This thesis will focus mainly on the third layer since it is there that the learning system is situated. This layer is probably the hardest one to implement since it is required to be:

- Adaptive: the layer must adjust itself to particular contexts that the user will visit.
- Transparent to the user: some user-interaction and training is required to enable the adaptivity.
- As autonomous as possible: the user supervision must be restricted to a minimum.

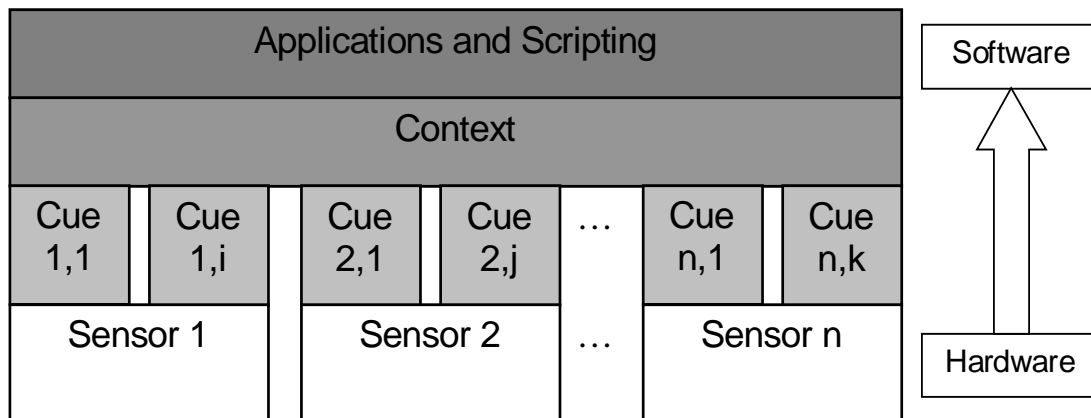


Figure 9: Layered architecture of the TEA project.

Currently, the sensor layer is the only one that has been integrated into hardware, while the other layers are implemented in software via a portable PC, connected to the hardware component. The objective is to eventually integrate all layers into hardware.

There is no sharp distinction between sensor data, cues, and contexts. In fact, it is very well possible to define the previous context as a sensor to obtain a recurrent structure in the architecture.

Chapter 3

Clustering the sensor data

As seen in the previous chapter, the values from all sensors are available to the system at any moment in time for processing, and these values can be treated as components of a –usually high dimensional- vector. If there were only three (or less) sensors present in the system, the outputs of these sensors could be plotted in a three-dimensional phase space by drawing a point with the values of the sensors as x-, y- and z-coordinates. The resulting phase space plot could be of some value for visualisation, but the limit of three sensors makes it practically not useful. It does illustrate however, the benefits and possibilities of clustering.

This chapter is divided in five parts:

1. A section on clusters in general supplies the necessary definitions of terms that are used in the remainder of this chapter.
2. Sensor-weighting and pre-processing are discussed in the second section to provide an efficient framework for the neural network section.
3. The third section gives a short introduction on traditional artificial neural networks, feed forward architectures and backpropagation.
4. The reason why the clustering will be done with an unsupervised neural network algorithm will be explained in the next section, along with the reasons why other, related approaches were not used.
5. Finally, experiments demonstrate how the clustering can contribute to context recognition.

3.1. Clustering algorithms in general

Fritzke [11] defines clustering as searching a partition of the data into subgroups, such that the distance of data items within the same cluster is small and the distance of data items stemming from different clusters is large. Many different kinds of clustering algorithms exist depending for instance on whether the number of clusters is pre-defined or not.

Two fundamental and closely related terms from computational geometry are important to understand in the context of clustering: Voronoi tessellation and

Delaunay Triangulation. The first concept is the partitioning of a space in Voronoi regions (hence the name tessellation), which are the sets of all points which are closest to the point each Voronoi region belongs to. If one connects all pairs of points for which the respective Voronoi regions share an edge, the Delaunay triangulation will be the result. Figure 10 represents a point set in \mathfrak{R}^2 , its corresponding Voronoi tessellation and Delaunay triangulation. These two expressions will be used later on in this chapter.

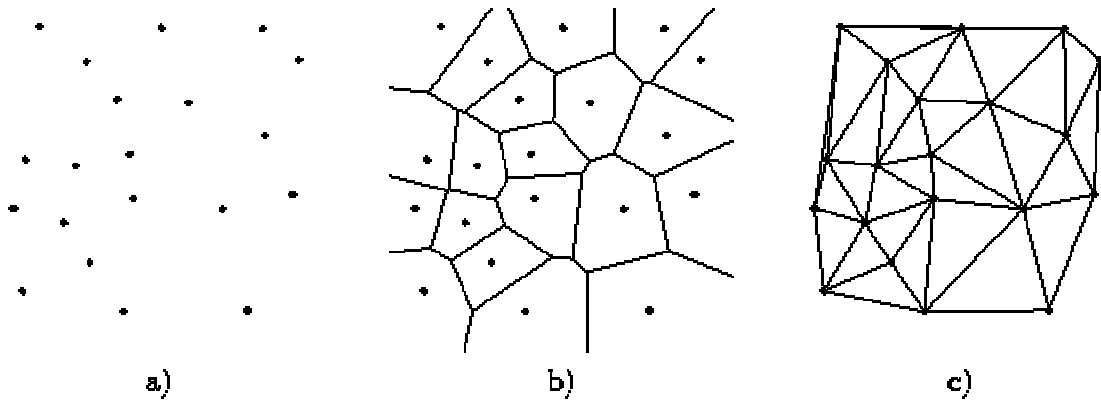


Figure 10: a) points in \mathfrak{R}^2 , b) corresponding Voronoi tessellation and c) corresponding Delaunay triangulation. (source: [11])

Using an unsupervised algorithm before the actual classification is – apart from a common thing to do in on-line learning applications – advantageous if the user is not full-time supervising the system. It would be even possible to divide the signals into preliminary anonymous classes that can be assigned a label by the user afterwards. Since the user-defined contexts are very general most of the time, a lot of these resulting clusters will usually be linked together to a single context profile. If for instance the context would be “in a dark room”, multiple clusters would emerge during the training phase that all correspond to that particular context. The two main reasons for this phenomenon are the general aspect of the context descriptions and the presence of redundant sensors for that specific context. This will result in more than one cluster emerging since redundant sensors can have a whole range of values for just one context, while the recognised context remains the same. Light sensors for example would be important to the “dark room” context, while other sensors like the temperature sensor or the microphone would be less important. The less (or not-) important sensors would then produce irrelevant inputs.

Clustering algorithms (and learning algorithms in general) can be divided in two classes: those who are ‘eager’ to learn, and those who are ‘lazy’. The first class wants to generalise before the user made any query, while the second one would delay the processing until a query is encountered. Lazy clustering algorithms are not very feasible for this purpose, since they generally require a lot of memory to store the training examples and more computation time when it needs to predict the target value for a new query. This could damage the real-time property of the learning system. The user query in this case is approximately every second, so it would be useless to postpone the clustering until a query is made. Another reason to choose for immediately clustering the incoming data is the high dimensionality (or the high number of sensors) of the input. The basic requirements of this thesis demand to implement an eager learning system, but this does not mean that some modules can be implemented by using lazy algorithms.

Every clustering algorithm works with a measure of similarity and dissimilarity to estimate if a certain pattern belongs to a cluster or not. Usually the Euclidean distance between the two vectors is used for this purpose.

One of the simplest algorithms is the *sequential leader algorithm*. Every cluster has a so-called prototype -or codebook vector, which is used for comparison with the current input vector, and which represents the prototype value of the weights of the cluster. If the codebook vector and the input vector are similar enough, the input vector belongs to the cluster of that codebook vector. In the case that no such cluster can be found, a new cluster is constructed with the input as its codebook vector. The leader-clustering algorithm is usually very fast: in the best case, all n inputs belong to the same cluster, resulting in time complexity n . A worst case scenario would result in every input being not similar enough to its predecessors, which will take $(n!)$ time steps.

-
1. Select the first input as the prototype vector for the first cluster.
 2. Compare the next input to the previous cluster prototypes

If the distance between the new input and a cluster prototype is less than a threshold:

cluster it with that cluster

otherwise add a new cluster with the new input as prototype

Repeat step 2 for all remaining inputs.

Table 1: The leader clustering algorithm.

Nevertheless, the leader clustering algorithm is not adaptive enough to use it on data coming from the low-level sensors. This data will always contain some noise, which will result in a poor performance of the algorithm.

Clustering algorithms were originally designed for handling discrete input vectors, but they can also be extended to work on time series. Various approaches exist that enable the clustering algorithms to compare time series, but the one that is most used is the calculation of the Euclidean distance between two time series:

$$dissimilarity = \|X_T - Y_T\| = \sqrt{\sum_i (x_i - y_i)^2}$$

where $X_T=(x_1, \dots, x_n)$ and $Y_T=(y_1, \dots, y_n)$ are the two time series for a certain sensor. This choice was inspired by the method of delays (described in [21] and [22]), based on theory about dynamics.

These approaches need to define a suitable time interval however, which is not very easy to determine. It should be long enough to hold specific time-related patterns, but if this interval is too large, the performance of the adaptive system will go down rapidly. The sensor outputs have usually a graph, which is too complex to be recognised by this method.

3.2. Sensor weighting and other pre-processing techniques

To optimise the system's ability to distinguish as many contexts as possible, a large variety of sensors is usually applied. This, however, adds a few difficulties since not all sensors are equally important and they do not necessarily have the same units and the same ranges of values. One solution could be attaching weights to each value, but the next difficulty will be finding a method to fix these weights in an appropriate way. Simple applications might rely on the user to handpick the weights, but this method becomes harder and harder when the dimension of the input gets higher. An algorithm might thus be required to find a suitable set of weights and adapt it if necessary.

Rosenstein and Cohen (in [23]) tackled this problem by first clustering all outputs of sensors individually and constructing a signature afterwards that enables unit- and scale-independent comparison. The signature stores the pattern of similarity between the new time series and the alphabet that was created in the clustering process.

After a clustering algorithm is chosen, another decision is to be made: should the input values be standardized, normalized, rescaled or maybe pre-processed in any other way? The difference with this type of pre-processing and the cues from chapter 2 is that the cue is calculated from the raw data that is omitted when everything is presented to the neural network. The calculations that will be discussed here will be applied on all sensors and cues without reducing the information itself.

A lot of clustering algorithms (including the SOM and RSOM discussed in the next section) depend highly on Euclidean distances. The contribution that an input makes will depend heavily on its variability relative to the other inputs. If one input has a range of zero to one, while another input has a range of zero to one million, the contribution of the first input to the total Euclidean distances will be negligible compared to the second input. In this case, it is essential to rescale the inputs so that their variability reflects their importance, or at least is not in inverse relation to their importance. It is very common to standardize each input to the same range or the same standard deviation if better prior information is not available. If some inputs are known to be more important than others, it may help to scale the inputs such that the more important ones have larger variances (see sensor weighting).

The values from the sensors are already limited between 0 and 127, so it is not actually necessary to pre-process the input values for the clustering to perform better. However, in these experiments not only the normalized sensor values are taken, but also cues like the mean and variance of the rapidly changing sensor values over the period of approximately a second (see the previous chapter). Therefore, all values are rescaled to the interval [0,1] to make things less complicated.

3.3. Neural networks

An artificial neural network refers to a (usually large) number of simple, interconnected computational elements: neurons. In this respect, it is based on the biological nervous systems. A typical neuron sums the weighted inputs and passes

the result of this calculation through an output link to another neuron. This result is often initially put through a so-called activation function, which is usually a non-linear function (e.g. a sigmoid) that limits the final output of the neuron. A bias is also included to provide a threshold that is subtracted from the sum of weighted inputs, although it could also be implemented as a connected neuron with a constant value. Activation functions in a hidden (or intermediate) layer are needed to introduce the possibility of solving non-linear problems into the network, while they are needed for the distribution of the target values in the output layer. Most neural networks have some sort of training rule whereby the weights of the connections are adjusted on the basis of input data. In other words, neural networks learn from examples and exhibit some capability for generalization beyond the data from the training phase. Another property of neural networks is their great potential for parallelism, since the computations of the neurons are largely independent of each other. Very good introductions about the theoretical aspects of artificial neural networks are given in [13], [17], [18] and [19].

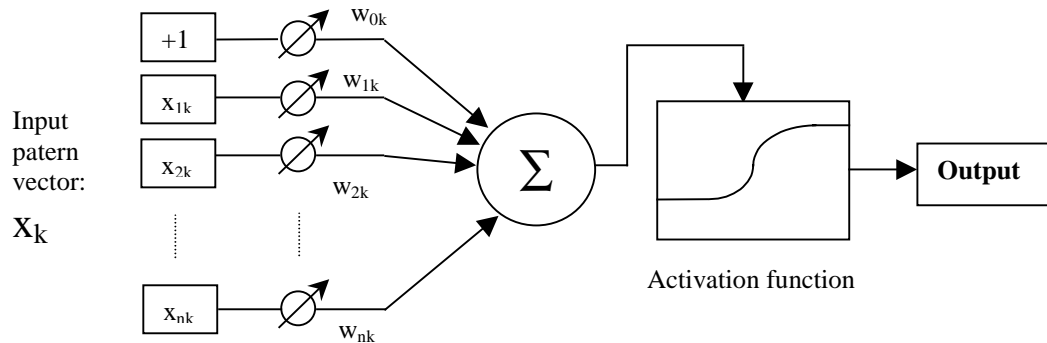


Figure 11: Structure of an artificial neuron.

One of the most popular neural networks is a feed-forward network with one or more hidden or intermediate layers and backpropagation as the learning rule. It is based on the gradient descent rule of the simple perceptron (without any hidden layer), which updates the weights by evaluating the error between the actual output and the expected output. If the network has one or more layers between the output and input layer, a similar method is applied, but this time the error propagates from the output layer back towards the input layer (hence the name backpropagation).

Connectionist methods are widely used in signal processing, certainly when the input signal reflects real-world data. The data from sensors have several properties

that make it very suitable for artificial neural networks. One of the biggest problems in signal processing is the presence of noise, which is inherent to the real-world data and this often causes the ‘classic’ algorithms to fail. Artificial neural networks are known to perform very good under these circumstances.

Apart from these benefits, some difficulties will remain when using neural networks. Especially the curse of dimensionality will be mentioned (and dealt with) later. The training phase might also be a problem if the objective of the learning system is to need as little intervention of the user as possible.

The classic multiple layer perceptron with backpropagation training needs a training phase, after which the network becomes fixed. The result of this method is that the user needs to frequently update the network by re-training it every time a context changes a bit. It would be better if the user had to indicate and train the contexts once, leaving the system to adapt itself further after this phase without needing supervision of the user. The multiple layer perceptron is also very slow in training and if the training contains some errors, the performance of the network will go down significantly. An unsupervised neural network that functions better in these domains will be thoroughly discussed in the next section.

3.4. Mapping with neural networks

A first approach would be to directly map the combined sensor data to a space of context profiles. Since it is initially not necessary to know any targets, an unsupervised neural network is used to cluster the input vectors. The resulting (discrete) output space then becomes a lookup table for the actual outcome. A time-sensitive variant of the Kohonen Self-Organizing Map - “the Recurrent Self-Organizing Map” - was chosen for this purpose.

3.4.1. The Kohonen Self-Organizing Map

The Kohonen Self-Organizing Map [15] (KSOM or just SOM) is an unsupervised neural network: this means it does not need any feedback from a teacher (or the environment as a teacher), unlike many other neural networks. The Kohonen SOM is not an ordinary neural network and the implementation of its neurons differs strongly from that of the usual neurons that were previously

described. The Kohonen network (1982) is based upon earlier work of Willshaw and von der Malsburg (1976).

The KSOM usually has an output layer of interconnected neurons that are fully connected to the input layer, so every neuron from the output layer is connected to every neuron from the input layer. This connection has a certain value, which is just like in the domain of the supervised networks called a weight. Every neuron from the output layer has consequently as many weights as the neural network has inputs. The output neurons are furthermore ordered in a particular way, usually a two-dimensional grid, where each neuron has ‘neighbours’ (to the left, the right, up and down in the case of a grid). The KSOM was inspired by the way in which various human sensory impressions are topographically mapped into the neurons of the brain. Figure 12 illustrates the structure of a SOM, while Figure 13 shows a typical organisation of the output layer of the KSOM; the colour of each square indicates during what context it was activated.

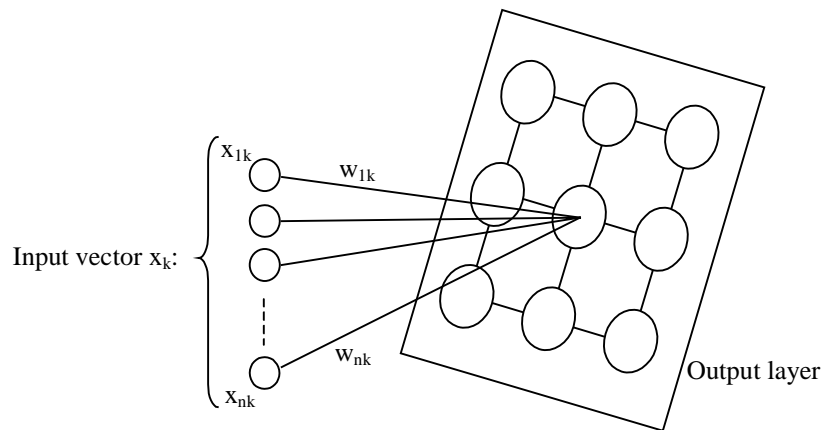


Figure 12: Structure of a Kohonen SOM.

Each time an input is presented to the SOM, this input vector is compared to the weight vector of every neuron from the output layer. The neuron with the most similar weight vector is selected and is permitted to update its weights towards the values of the input vector. This is why the output layer is also often called the competitive layer: each neuron ‘fights’ over the input and the winner is granted an update of its weights. If a similar input is presented afterwards, this neuron will be more likely to win again because it could update its weights while the weights of the other neurons remained unchanged. The formula to find a winner is consequently given by:

$$\min_{w_{ij}} (\|x - w_{ij}\|) \quad \text{for all weights } w_{ij} \text{ of the neurons on the position } (i,j)$$

To introduce order in the output layer, this algorithm is modified in such a way that the neighbouring neurons of the winner are also allowed to update their weights, but not as much as the winner itself:

$$w_i(t+1) = w_i(t) + \alpha \eta(t) (x_i(t) - w_i(t))$$

where α is the learning rate and η depends on the distance to the winner.

This algorithm is responsible for creating and adapting neurons that are trained to trigger for a specific input: the values from the high dimensional input space are mapped to a discrete output space (the grid of neurons). The weights of the output neurons are very similar to the codebook vectors of the leader clustering algorithm, but in this case they are frequently adapted. It is thus very well possible that the weights are different from all previous input vectors. Another big difference between the two algorithms is that the SOM has a fixed structure that does not grow in time, while the leader-clustering algorithm could theoretically keep on adding clusters.

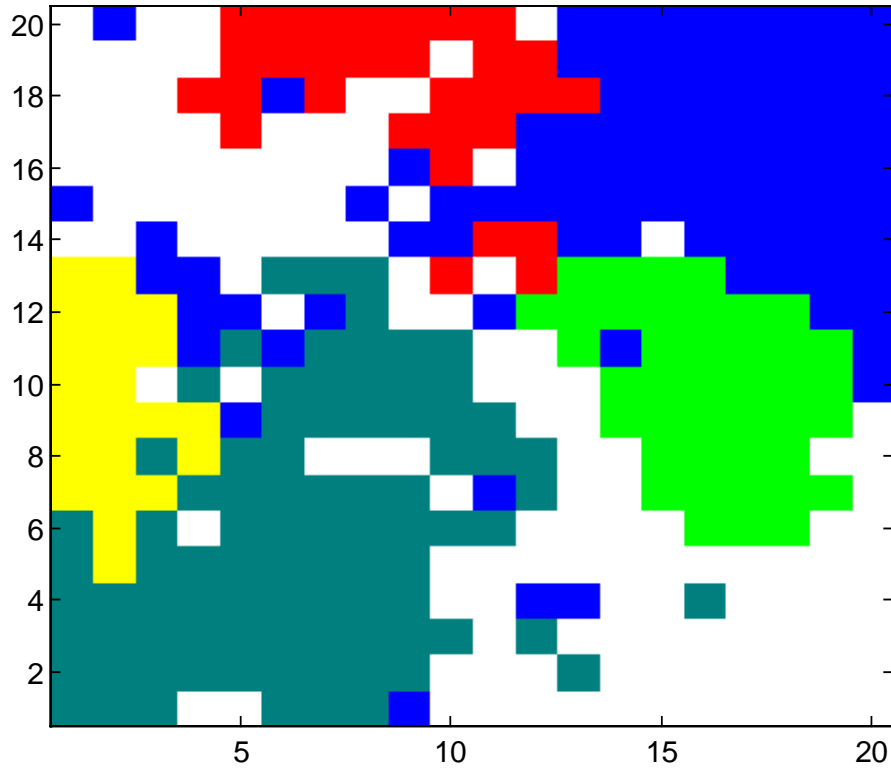


Figure 13: Example of the output layer of a SOM.

The KSOM (like most of its variants) has limits that have a bigger impact on its performance, though. The inputs need to remain between a minimal and maximum value for the clustering to be successful and reliable. If the input has no upper and lower limits, locations of the clusters could keep on changing in time. Another well-known problem of the Kohonen network is that it is possible it will unlearn previous learned clusters, certainly if the learning rate is chosen relatively high. It is very common to implement the KSOM so that the learning rate and the neighbourhood radius get gradually smaller until they reach a certain minimal value, resulting in a ‘frozen’ status. It is not a very good idea to diminish the learning rate or the neighbourhood radius to zero, however, because this means that the neural network will lose its adaptive behaviour. Using extremely small learning rate and neighbourhood values is also no solution since this may critically hinder the training and thus the adaptivity. See section 3.4.3. (Stability versus Plasticity) for an elaboration on this subject.

-
1. Initialize the weights from N inputs to M outputs neurons to small random values.
 2. Present a new N dimensional input vector $x_i(t)$ to the neural network
 3. Calculate the distances between this input vector and the weights $w_i(t)$ of every neuron, using the Euclidean distance.
 4. Find the minimal distance and select the corresponding neuron as the winner
 5. Update the weights of the winner towards the input:
 $w_i(t+1) = w_i(t) + \alpha \cdot (x_i(t) - w_i(t))$ with α the learning rate between 0 and 1.
 6. Update the weights of the neighbours, depending on the distance to the winner:
 $w_i(t+1) = w_i(t) + \alpha \cdot \eta(t) \cdot (x_i(t) - w_i(t))$ with $\eta \in]0,1[$ the neighbourhood function which decreases when the neurons get further away from the winner.
 7. Go to step 2
-

Table 2: The algorithm to produce Kohonen SOMs.

The algorithm in Table 2 contains the necessary steps to produce a Kohonen Self-Organizing Map. The neighbourhood function has usually a bell-shaped curve. A common choice for this function would be

$$\exp\left(-\frac{\|r_i - r_c\|^2}{\sigma^2}\right),$$

with r_i and r_c the coordinate vectors of the current neuron and the winning neuron. The function plot is bell-shaped around r_c with a width depending on σ .

Step 6 in the algorithm is needed to order the codebook vectors of the neurons topologically, and transforms a normal competitive neural network into the Kohonen SOM. Some discussion exists about the necessity of the presence of topology-preserving abilities in the algorithm, but it has benefits as will be shown later.

The Kohonen SOM has on the other hand also a lot to offer. Visualisation is one of the reasons why it is used frequently. Apart from the graphs that represent the output layer, there is another kind of plot that gives more information about the network's point of view - this method will be explained at the end of this section.

3.4.2. Curse of Dimensionality

The potentially high dimensionality of the data is a problem with this approach. As more sensors are added to the system, the dimension of the data gets higher, and the clustering algorithm needs more resources to be able to map the bigger input space to the output space. Unless the neural network is improved (by for example adding a – usually substantial – amount of neurons), the consequences are that the algorithm gets slower and less fault tolerant. What causes this effect? Suppose that the SOM is able to place the weights of the output neurons equally over the input space. The average distance from a random point of the input space to the nearest weight vector could be measured as the goodness of the representation: the shorter the distance, the better is the representation of the data. The total number of units required to keep the average distance constant increases exponentially with the dimensionality of the input space. This also causes networks with lots of irrelevant inputs to behave relatively badly: the high dimension of the input space causes the network to use almost all its resources to represent irrelevant portions of the space. This problem is well known in the domain of neural networks (or even more generally in the field of machine learning) as the “Curse of Dimensionality” (see [5] and [19]).

3.4.3. Stability versus Plasticity

One of the major problems with KSOMs is to keep the map adaptive without ‘overwriting’ already learned instances. If the system is created to remain adaptive (which usually means it has a fixed, nonzero learning and neighbourhood radius) its neurons could gradually change to other clusters, thereby unlearning previous stored clusters. If, on the other hand, the KSOM’s learning rate and neighbourhood radius diminish over time, the system will converge to a stable one without unlearning. This method is suggested by Kohonen [15], but an immediate result of this is that the system will end up in a non-adaptive state. This trade-off feature is called the stability-plasticity dilemma.

A simple and common method to overcome this problem is tweaking the learning rate and neighbourhood radius so that the overwriting becomes very unlikely. In the initial phase, it is still possible to present various contexts with a larger learning rate and neighbourhood to reach an optimal spreading of the weights of the competitive layer and to avoid “dead regions” on the competitive layer.

In section 3.4.6., the adaptive resonance theory will be discussed which is said to actually solve the stability-plasticity dilemma.

3.4.4. Temporal variations on Kohonen's SOM

The input data from the sensors doesn't just consist of vectors that can be presented at random to the KSOM network, the sequence in which they appear is equally important. The Kohonen Self-Organizing Map in its standard form is not able to distinguish time series. However, a number of approaches exist to improve the KSOM in order to enable it to recognise time series.

One approach is to add temporal information to the input of the network by means of exponential averaging the input time series or using tapped-delay lines. Another method is to use layered or hierarchical KSOMs where the upper map needs to capture the spatial dynamics of the previous layer. Euliano and Principe modified the KSOM to a Spatio-Temporal Self-Organizing Feature Map [10], which uses temporally and spatially localized neighbourhoods. Each time a node wins, it starts waves of activity that flow over the output layer and which are attenuated over time. These waves can reinforce each other if they correlate temporally. The activity wavefronts are used to enhance a node's possibility of winning the next competition.

The Temporal Kohonen Map (TKM) uses output neurons that do not immediately restore their values to zero, but that gradually lose activity. Another approach - based on the TKM - is to add a context layer to the SOM's input architecture, enabling it to memorize past inputs, similar to the supervised Elman or Jordan neural networks. Each neuron has, apart from a weight vector, a recurrent difference vector, which keeps track of past input vectors. The map is able to store information about the change of the input vector, and it can thus cluster the time series temporally. The resulting network, the Recurrent Self-Organizing Map (RSOM) [16] [14] [29], is able to learn time-varying patterns and has proved valuable in various studies.

The RSOM uses *difference vectors* that are associated to each neuron in the network. The difference vector $y_i(t)$ of neuron i of the map V_M is updated as follows, with $x(t)$ the given input:

$$y_i(t) = (1 - \alpha)y_i(t-1) + \alpha x[(t) - w_i(t)]$$

where $w_i(t)$ is the weight vector of neuron i at time t and $\alpha \in]0,1[$ is the weighting factor (WF). The adaptation of the weight vectors is changed into:

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

where $i \in V_M$, γ is the learning rate and h_{ib} is the neighbourhood function.

3.4.5. Tracking the activation

When keeping track of the number of activations per neuron, the result will be a landscape-like surface with the exact same size of the competitive layer. The activation landscape proves to be an efficient visualisation method, which gives a lot of information about the clustering – not just the size and position of clusters, but also the density becomes visible. The main reason why hills are formed on this surface, is the topology-preserving ability of the Kohonen Map.

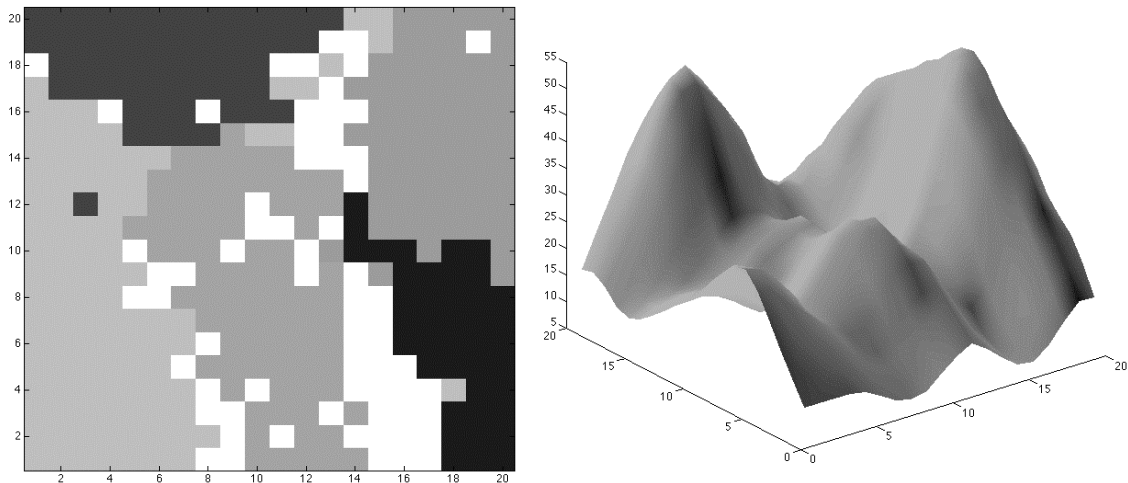


Figure 14 a: The SOM output layer. **b:** The corresponding activation surface.

In Figure 14a, the competitive layer of an already trained SOM is depicted, while the activation surface for the same map is plotted in Figure 14b. Four clusters are located at the edges of the SOM, one is located in the middle. The right side of the map is divided in two clusters, although these are more or less merged in one hill in the surface plot. It is also clearly visible in Figure 14b that the middle cluster is a combination of two or three hills. When the sensor outputs are mapped onto this

surface, one could deduce what the probability is that the learning system will be able to classify this. If it is mapped in a valley, it will be very likely that the learning system will have difficulties in establishing correct context recognition.

It is possible to mark the response of the Self-Organizing Map to the current input on this activation surface or activation landscape. A problem is that this surface will still adapt itself during the time these inputs are fed to the network. If the adaptive behaviour of the SOM is frozen (by making the learning rate equal to zero) after a first pass of the data however, the response of the SOM in a second pass can be visualized by drawing marks on the surface (as depicted in Figure 15).

Since the height of every x-y position is determined by the number of times the neuron on those coordinates has been activated, the hills or bubbles on the activation surface plot represent clusters of neurons which all respond to a similar input. In fact, it would be more correct to call the bubbles on the surface clusters of clusters, since every discrete position on the landscape represents a neuron of the KSOM's competitive layer, which is in fact already a cluster. The next step in the recognition process could be described as labelling the hills on the activation plot, in stead of putting a label on every discrete position (neuron). The latter method would generally need more processing time and memory. This classification will be discussed in chapters 4 and 5.

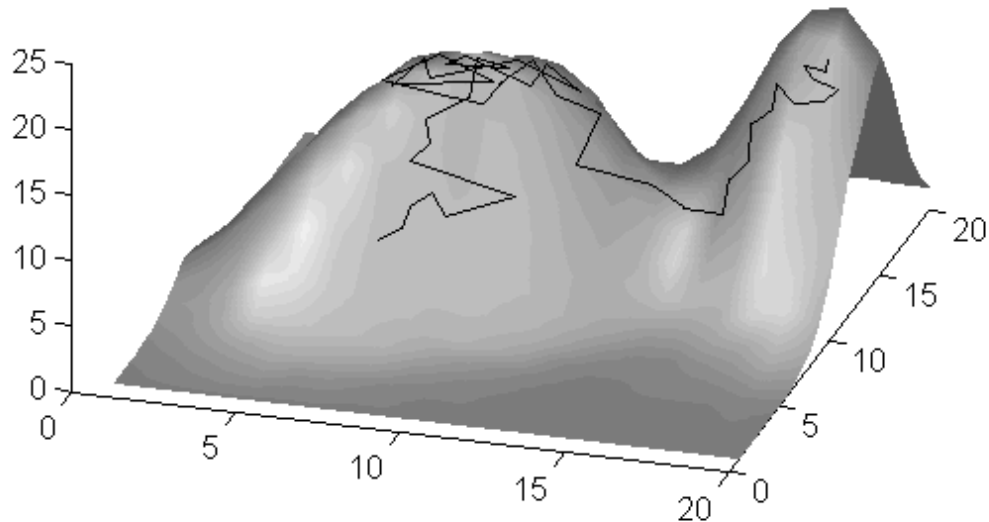


Figure 15: Tracking the input while walking from the inside to the outside context.

The activation landscape is an excellent way to track the winning neurons and visualise what is happening, comparable to the phase space graph discussed in chapter 2. This time, however, there is no limit in the number of sensors (or input dimension) since the input dimension has already been reduced by the Kohonen Self-Organizing Map.

3.4.6. The Adaptive Resonance Theory

Carpenter and Grossberg solve the stability-plasticity dilemma in their Adaptive Resonance Theory (ART) [6] by introducing top-down mechanisms that enable the system to stabilise itself. They designed a network which forms clusters and is trained without supervision: the Carpenter/Grossberg classifier [17]. The architecture embodies (like the SOM) a parallel search scheme, which updates itself adaptively as the learning process unfolds. After the learning self-stabilizes, the search process is automatically disengaged, and input patterns directly access their recognition codes without any search. The consequence is that recognition time does not grow as a function of code complexity. The architecture possesses a self-scaling property, which enables its codebook vectors (or ‘prototypes’ in the vocabulary of

ART) to form. They detect and memorize predictive configurations that are derived from the set of all previous input patterns.

To keep the system stable yet plastic, ART has some variables - priming, gain control and vigilance - build in the architecture. Priming and gain control are needed for code matching and self-stabilization, while vigilance determines how fine the learned categories will be. If the vigilance increases, then the system will automatically search for finer learning categories. Vigilance is actually a threshold, that determines how close a new input pattern must be to a stored instance to be considered similar. It is usually kept in a range between 0 and 1: a value near 1 requires a close match and smaller values accept a poorer match.

ART comes in several flavours, supervised as well as unsupervised. The unsupervised ARTs are basically similar to many iterative clustering algorithms (such as the leader-clustering algorithm discussed in the beginning of this chapter) in which each case is processed by these steps:

1. finding the "nearest" cluster seed (or prototype or codebook vector) to that case
2. updating that cluster seed to be "closer" to the case

where "nearest" and "closer" can be defined in many different ways. In ART, the framework is modified slightly by introducing the concept of "resonance" so that each case is processed by:

1. finding the "nearest" cluster seed that "resonates" with the case
2. updating that cluster seed to be "closer" to the case

"Resonance" is just a matter of being within a certain threshold of a second similarity measure. A crucial feature of ART is that if no seed resonates with the case, a new cluster is created as in the leader-clustering algorithm. As a consequence, the number of clusters will grow in time, depending on both the threshold and the distance metric used to implement similarity. The feature of adding a cluster is said to solve the "stability-plasticity dilemma".

The current training case is stored in a short-term memory (STM) and cluster codebook vectors are part of "long term memory". The two stages of finding the nearest seed to the input are performed by an Attentional Subsystem and an Orienting Subsystem. The latter of which performs "hypothesis testing", which refers

to the comparison with the vigilance threshold. This does not refer to hypothesis testing in the statistical sense. The ART learning method is stable, which means that the algorithm converges; it does not mean however that the clusters are insensitive to the sequence in which the training patterns are presented.

The ART network can perform well with perfect input patterns, but even a small amount of noise can cause problems. Due to the noise, the level of vigilance can be set too high resulting in a high number of stored instances, which can rapidly grow until all available nodes or memory is used up. Modifications to this algorithm are necessary to perform the clustering in this application since noise will always be present in the output of the sensors. These could include adapting weights more slowly and changing the vigilance threshold during training and testing, rendering the system either very slow, or very complex, and this is why the ART network was not used as the clustering algorithm. It might be worthwhile keeping in mind how it overcomes the stability-plasticity dilemma, though.

3.4.7. Other unsupervised neural networks

A large variety of unsupervised neural networks exists. A compact list of some of the most prominent kinds of unsupervised neural networks is included in this section plus the reason why they were not considered in this thesis.

- LBG:

The LBG (or generalized Lloyd) algorithm repeatedly moves the codebook vectors to the mean of each Voronoi region (discussed earlier in this chapter). It has been proven to converge in a finite number of iterations to a local minimum.

LBG is a batch method where all possible input signals (which must come from a finite set) are evaluated before any adaptation is done. The large input space in this thesis makes the LBG not feasible as a clustering algorithm.

- Neural Gas:

This algorithm first sorts all neurons of the network according to their weights' distance to the input vector. Based on this order, a certain number of neurons is adapted. Both the number of selected neurons and the adaptation strength are decreased in time.

The neural gas algorithm is very similar to the Kohonen SOM, but there is one difference between the two: topology. The KSOM has – unlike the neural gas - the ability to organize the output layer space. This feature is (apart from visualization) not necessary in every application, though. Many applications use the Kohonen SOM as a clustering algorithm without actually using the topology preserving property.

The decreasing adaptation properties also prevent the neural gas from remaining both stable and plastic (cfr. the stability-plasticity dilemma discussed earlier), which is needed for the thesis.

- (Stable) Growing Neural Gas:

This method has a variable (mostly increasing) number of neurons, which is changed during the self-organizing process. New neurons are inserted near the neurons that accumulate most errors to obtain a successful growth mechanism.

Sagaert [24] developed a stable version of this algorithm.

3.4. Experiments and results

In the first experiment the objective is to test the robustness of the context classification when using adaptive cue to context mapping, based on the Kohonen SOM network. In a training phase, data was collected while taking the TEA board through a sequence of contexts, related to “being inside in the hand”, “being inside on the table”, “being inside in a suitcase”, “being outside on a table” or “being outside in the hand”. Acquiring data from a sequence of contexts has the advantage that the experiment will be able to demonstrate not only the clustering of a context profile, but also the sensitivity to distinguish contexts. The KSOM clustering algorithm successfully mapped sensor readings into a two-dimensional grid in which areas of high activation correspond to these contexts. Figures 16a till 16e show the clustering during training of the self-organizing map for five passes through the training data; one pass (or *epoch*) is recorded data from all five contexts. This could be viewed as training the Kohonen network by putting the device in the five contexts, and this for five times in a row. Each iteration consists out of approximately five times 2 minutes per context (transitions were not counted).

It should be pointed out that parameter settings in the clustering algorithm (i.e. neighbourhood radius and learning rate) sometimes lead to the creation of spurious clusters, but in general the parameters do not need time-demanding fine tuning. The initial (random) weights are very important to the topology of the KSOM, but this feature is actually not necessary for the clustering itself. For the visualisation, however, this could make a difference. The labelling of these clusters in this experiment was added by hand, but in a next chapter, this will be done in a more realistic, feasible and automated way.

After this first phase, the recognition of these contexts was tested when taking the board through similar (but not identical) context changes. In informal evaluation, a robust classification can be observed when attempting to follow similar contexts as the ones in the training set.

Figures 17a to 17e show the tracking of the inputs of the second phase on top of the ‘frozen’ surface from the first phase.

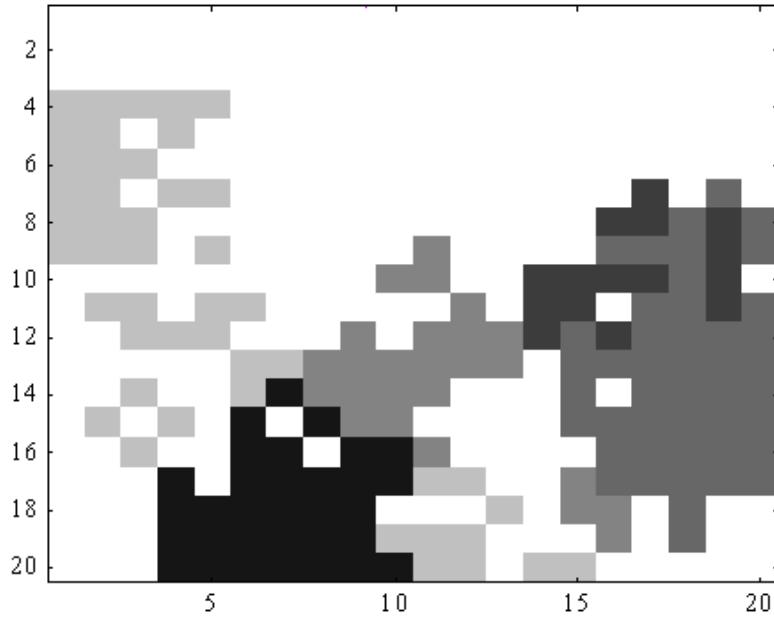


Figure 16a: The output layer of the KSOM after one epoch.

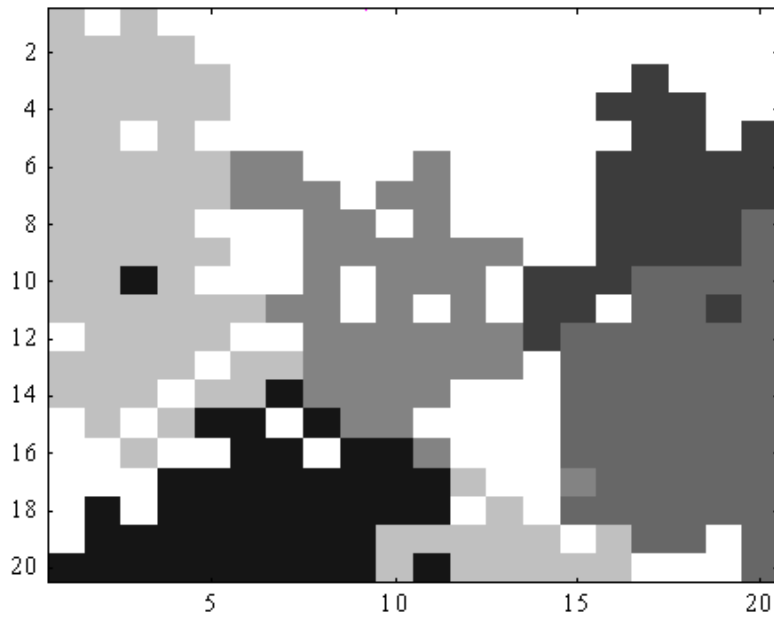


Figure 16b: The output layer of the KSOM after two epochs.

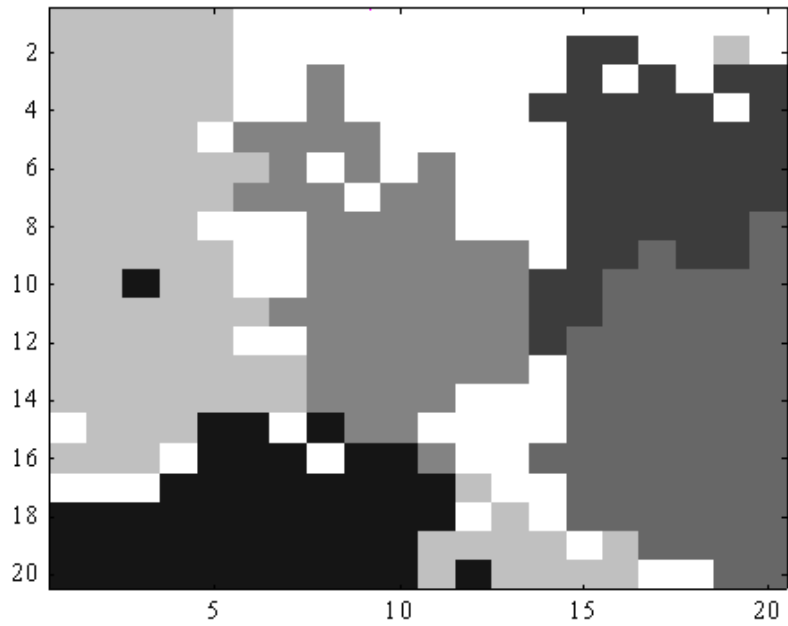


Figure 16c: The output layer of the KSOM after three epochs.

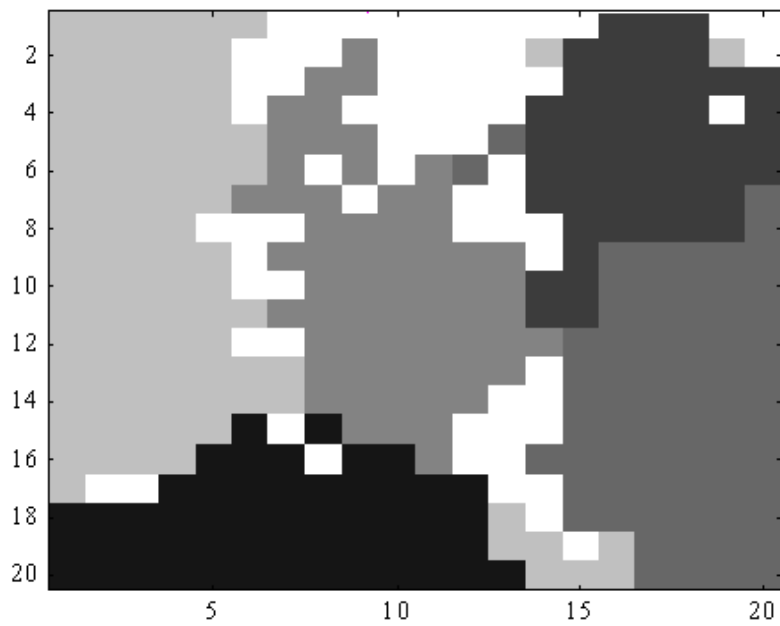


Figure 16d: The output layer of the KSOM after four epochs.

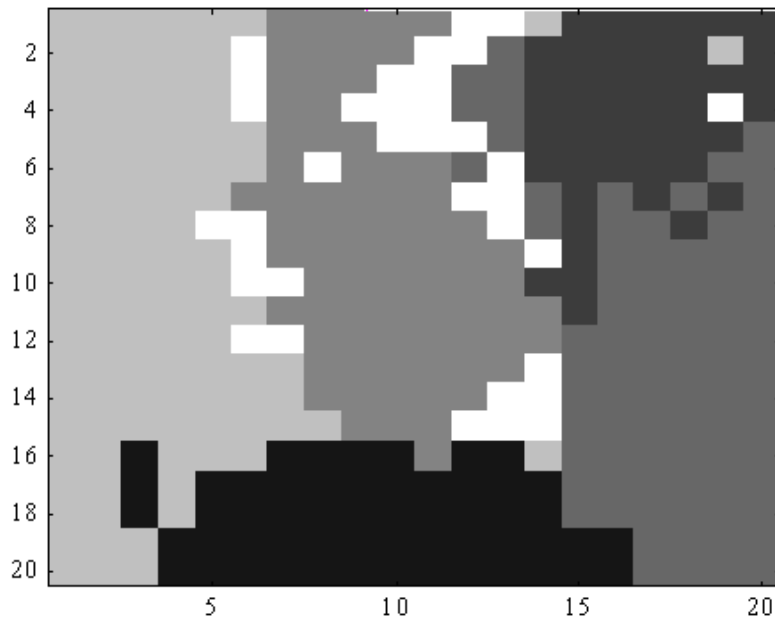
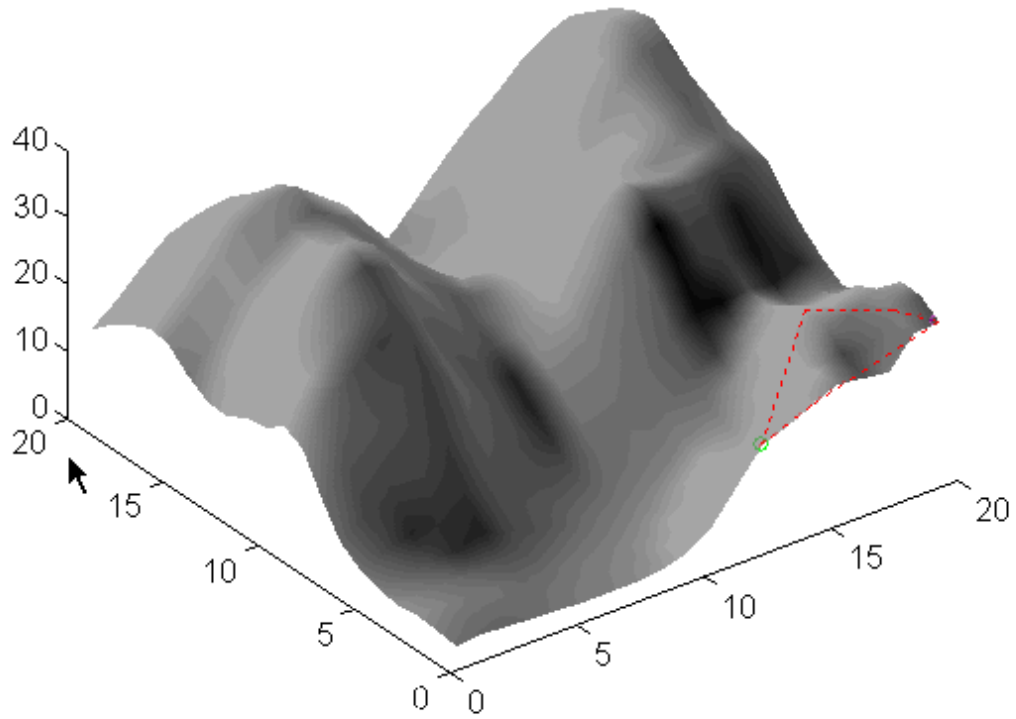
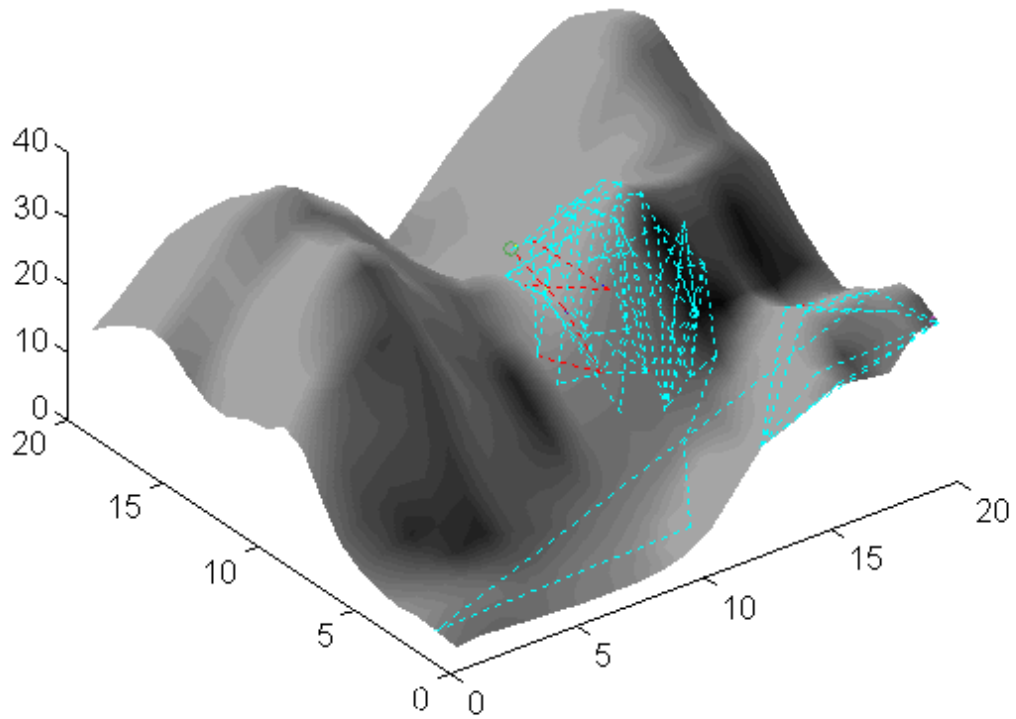


Figure 16e: The output layer of the KSOM after 5 epochs.



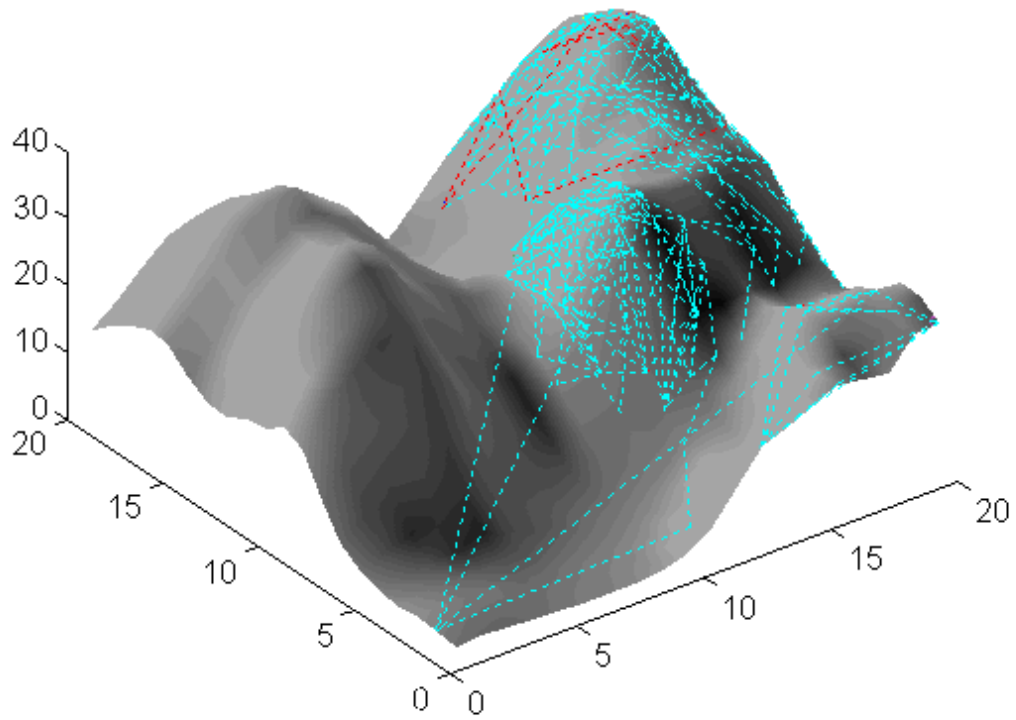
1000

Figure 17a: Tracking the input on a frozen activation surface during the first context.



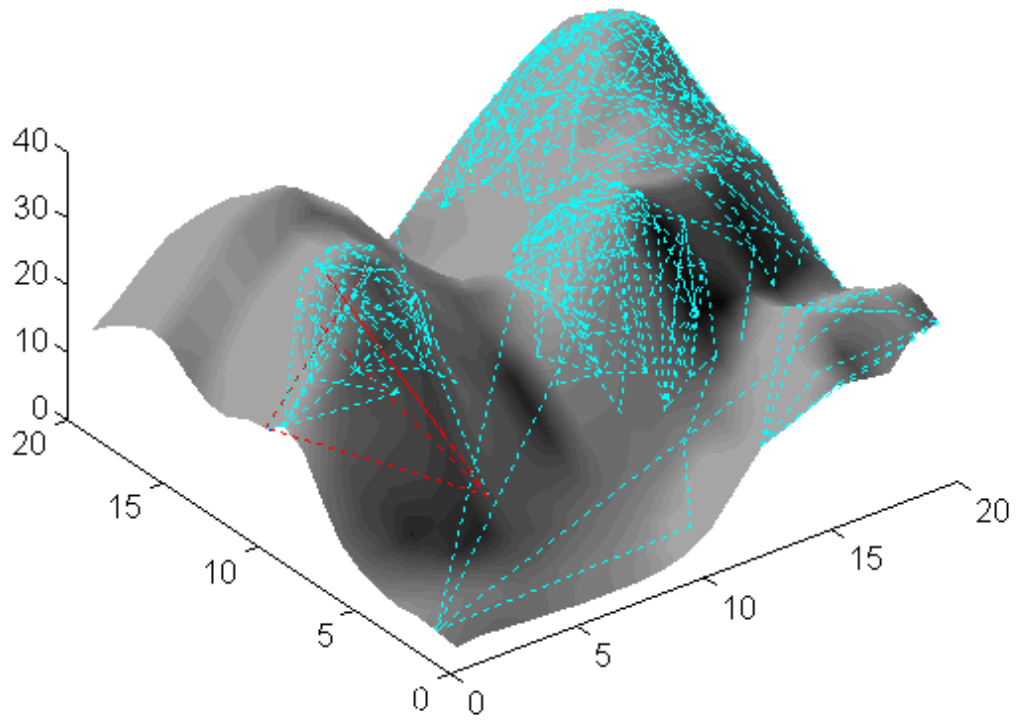
265

Figure 17b: Tracking the input on a frozen activation surface during a second context.



448

Figure 17c: Tracking the input on a frozen activation surface during a third context.



640

Figure 17d: Tracking the input on a frozen activation surface during the fourth context.

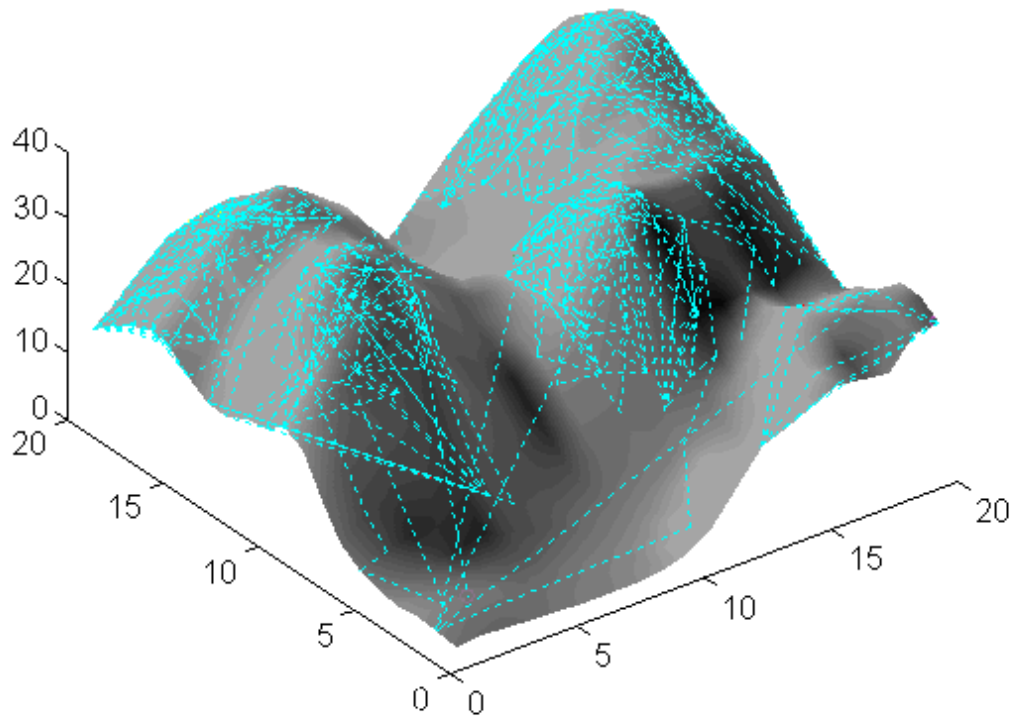


Figure 17e: Finished tracking the inputs on an activation surface during five contexts.

Chapter 4

The intermediate layer: dividing the input space

When the number of sensors increases, the size and complexity of the Self-Organizing Map (and a lot of other clustering algorithms) might become too large for an on-line application. In that case, it is possible to use multiple smaller SOMs for only subsets of the total set of sensors. It is trivial that the choice of the subsets will be crucial to the final performance of the clustering.

4.1. Dealing with high dimensionality

To evade the curse of dimensionality, a large variety of approaches exists, but one of the most effective ones is by simply reducing the dimensionality of the problem by splitting the inputs up into smaller segments. If the subsets are chosen in an appropriate way, the combined processing resources needed for these smaller inputs will be far less than the resources needed by the initial system. This divide-and-conquer technique will usually need an additional processing layer to combine the results of processing the subsets of the input data. In this particular case, the solution is to use intermediate mappings and to exclude sensors for each mapping. Rather than using one big neural network that needs all sensor readings as input, multiple smaller neural networks are used that start from a subset of the set of all sensors.

There is no limit in the number of subsets that have their own clustering algorithm, but using multiple algorithms in parallel can be inopportune in the event that the processing must be done in soft real-time. A trade-off can be observed between clustering precision and necessary resources to obtain real-time processing. This on-line specification is mainly necessary to limit the amount of user involvement during the use of the system.

The possibility exists to use more than one kind of SOM or even more than one kind of clustering technique in parallel. One can for instance choose to cluster sensors that are not frequently being updated with the regular SOM, while the RSOM or a similar algorithm could process data that possibly contains time dependent

patterns. Again, the choice of these algorithms can be very important to the overall performance of the system.

An additional benefit of this approach is that it is not necessary to use very large output layers, since there will be generally less clusters in the output space. Reducing both the size of the input vectors and the size of the output layers should lead to a faster and more precise creation and adaptation of the clusters. These clusters are, however, numerous compared to those of the initial approach, and it might be necessary to – again – use a clustering algorithm if the output space of the first layer is too large.

This method results in a change in the output of the clustering layer, since there are more than one output layers (or two-dimensional cluster spaces). When using just one KSOM, the result of the layer is equal to the positions of one or more neurons in the output layer. In this new approach the output is made out of positions in multiple output layers, possibly too many to handle directly. In that case a second clustering algorithm could prove useful – a smaller and simpler one, like the k-nearest neighbour, is preferred for this purpose in the experiments.

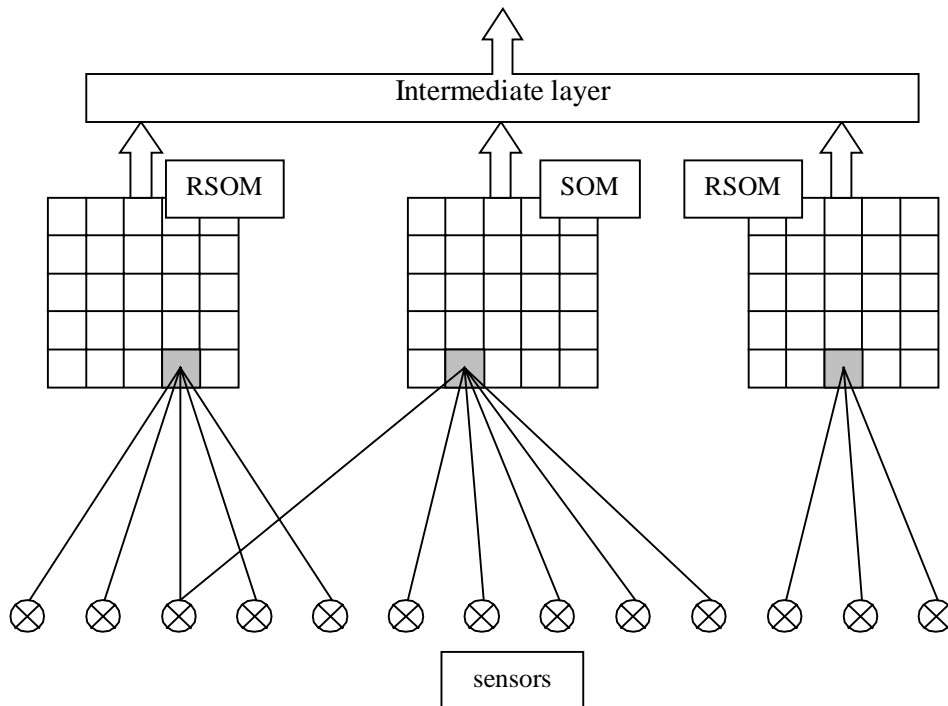


Figure 18: An example of the architecture of the system so far.

An imaginary implementation of this approach is illustrated in Figure 18. Notice that it is possible (and feasible) to use the output of a sensor for more than one sub-clustering algorithm. The weights are only drawn for one neuron in the grid per clustering algorithm to enhance the visualisation, normally every neuron should have links to these sensors.

A trade-off exists between having a slow, bulky algorithm that is able to combine all sensor outputs, and having several small and fast algorithms that omit (possibly relevant) combinations.

4.2. Experiments and results

The next experiment will demonstrate the usefulness of dividing the KSOM into smaller ones when the input space grows larger. The original clustering layer has one RSOM with an output layer of 400 neurons (20x20 grid), while the second version has two smaller ones – one KSOM and one RSOM – both with an output layer of 100 (10x10) neurons. The shrinking of the number of neurons is justified because the number of clusters in the output space will be less as explained in the previous section. The output layer is not the only element that becomes smaller, the weights attached to each neuron become smaller as well since the input space has become smaller. The output layer is reduced from 400 (20x20) neurons to 200 neurons (2x10x10). The total number of weights becomes thus 3200 (8x400) for the first architecture, while it is 800 (2x4x100) for the second. The outcome is a faster algorithm that is also less demanding on the hardware resources. Table 3 gives a summary on the comparison of both systems in terms of numbers.

Architecture:	One giant RSOM	Two smaller SOMs
Output layer:	400(=20x20)	200 (= 100 (=10x10) + 100 (=10x10))
Inputs:	8	4 + 4
Total weights:	3200	800

Table 3: Facts and Figures of the experiment.

The choice of the subset is done more or less randomly for all sensors but the light-related and motion-related sensors and cues were kept together in the RSOM (since they show short time-varying patterns). The WF threshold of the RSOM was kept close to one, since a longer-term memory was not really needed in this case.

The next two graphs (Figures 19 and 20) show the results in recognition, with a minimum amount of training. The upper plot shows the ID given by the clustering-classification layers. Valid IDs are 1, 2, 3, 4 and 5, while zero was the ID to represent the ‘not recognized’ outputs. The fact that the classification fails frequently is the result of a minimal training (a few seconds per context). This will provide a good view on the responsibilities and qualities of the supervising layer, described in chapter 6. Both the supervising layer and the classification layer are discussed in following chapters, so they will be explained later. It is now sufficient to look at the lower plots to see the improvements when using the two smaller SOMs. The function in the lower plot depicts the context ID that was given by the supervising layer. This would be the actual result of the whole system if the ID number were labelled to a short description like “jogging in the park”. The vertical lines through both plots represent the actual (added by hand) borders between the five contexts.

The results of the divided input space approach are a bit better than those of the one clustering algorithm method. Especially the transition from the first to the second context becomes better, while the others are less visible. Notice that the results from the classification layer are also in general better. This of course does not prove that results become better in every case; the experiment is a very extreme example since the amount of training was intentionally kept very low.

Dividing the sensors into groups results into omitting potential combinations of sensors, that are useful in distinguishing contexts. It is thus possible that dividing the input space has negative effects on the quality of the clustering. The grouping of the sensors should consequently be chosen very carefully, and overlapping (one sensor or cue present in two groups) should be considered. Omitting combinations can have either a good effect or a bad effect, depending on whether these combinations are respectively relevant or not.

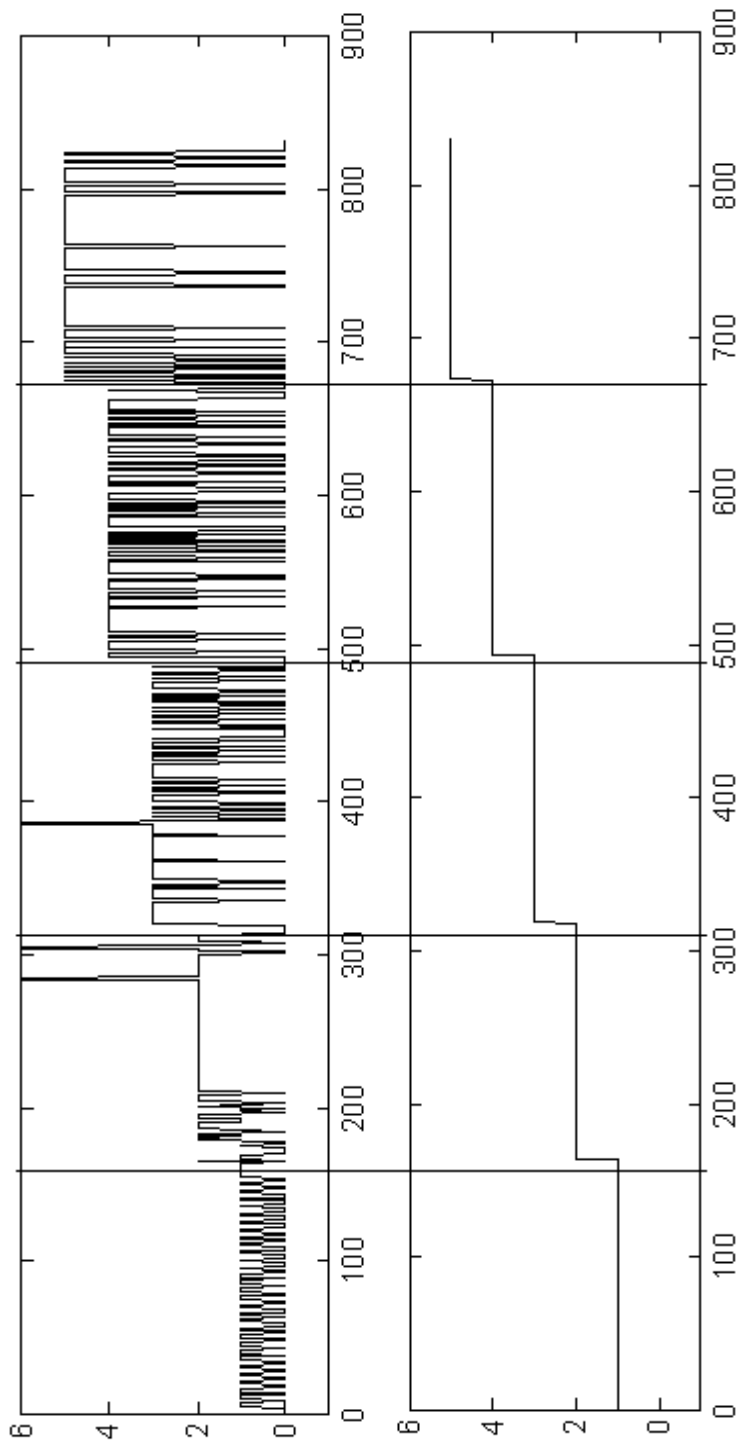


Figure 19 : Recognition of contexts with two SOMs in parallel

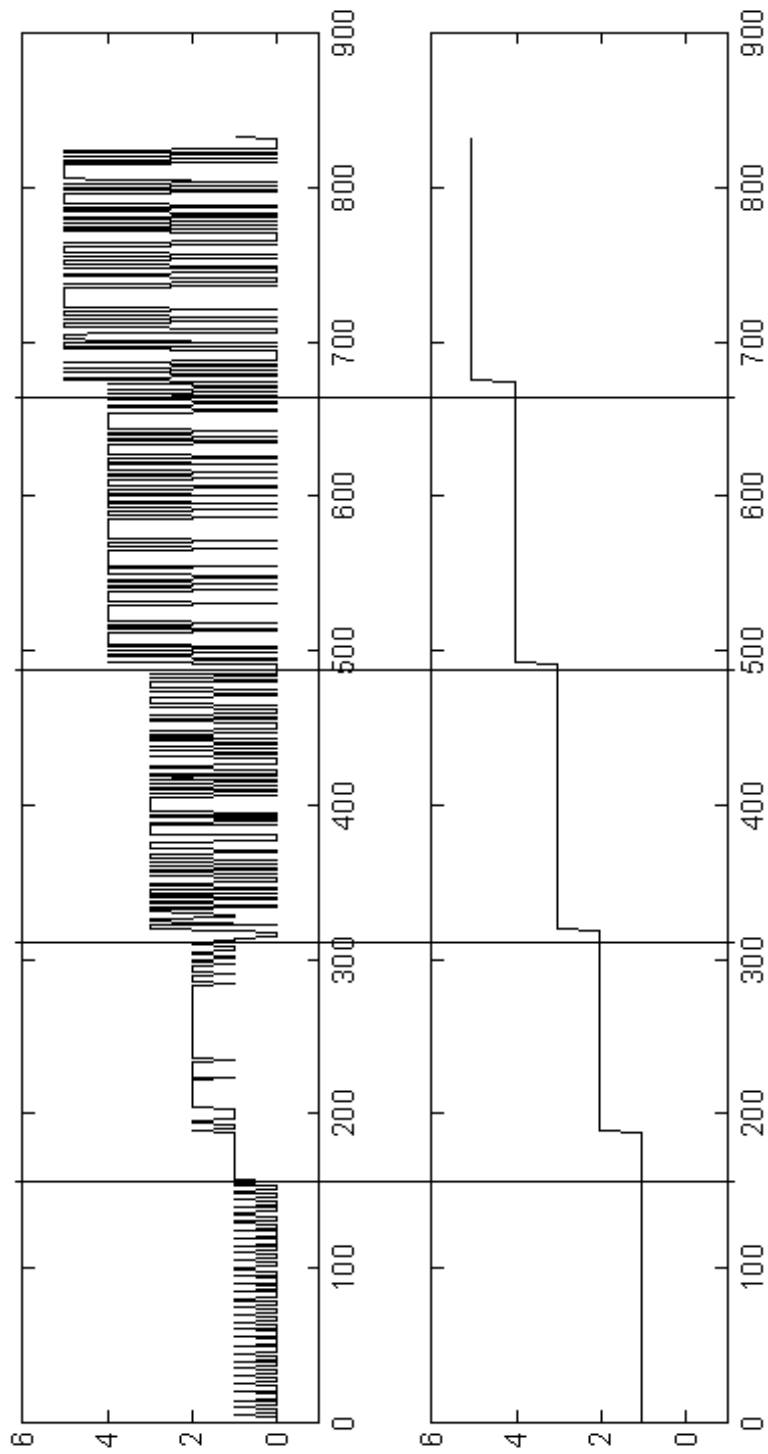


Figure 20 : Recognition of contexts with one big SOM

Chapter 5

Classification algorithms

The work is not entirely done after the clustering algorithm has finished processing the input. The clustered input vector still needs to be associated with a context profile, in order to complete a classification of the original sensor readings. The goal of this chapter is to look for a good classification method that can be used in the strategy of the thesis. This chapter elaborates on some classification algorithms that could be used. Another problem that will be discussed in this chapter is how the system can be trained to recognise context profiles in an easy way without the necessity of the user being too much involved in training.

5.1. Simple association

Every neuron in the SOM can be linked to a context profile if the number and size(s) of SOMs is limited. In general, every cluster that was created by the clustering algorithm will be given a label. This method was determined to work best with the experiments in this thesis, since both the number of classes and the size of output layer during the experiments was rather low. A complete overview of the algorithms that can be implemented is given in this chapter to anticipate the growing amount of sensors.

This brute force method could result in a slow system if the number of output neurons grows too high. One solution would be to cluster the neurons (which are in a way already clusters) again and attach a context profile to each new cluster. This method is in fact similar to the nearest neighbour algorithm in the next section.

5.2. Distance Weighted K Nearest Neighbours

Once the clusters are created and linked to a context profile, new values coming from the (R) SOM have to be classified, preferably by a fast but accurate algorithm. The nearest neighbour algorithm is one of the simplest and fastest algorithms: it compares the resulting output (an activated cluster in this case) to a set of training instances and classifies it to the most similar instance. Normally the training data consists of a set of instances plus the classes they should be associated

with; in this case the training data is created from the clustered codebook vectors of the SOM's neurons.

For approximating the discrete function $f : \mathcal{X}^n \rightarrow V$

Training algorithm:

Add each training instance to a list of instances, `training_list`.

Classification algorithm:

Given an instance x_q to be classified:

Let x_1, \dots, x_k denote the k instances from the `training_list` that are nearest to x_q :

return $f(x_q) = \arg \max_{v \in V} \sum_i w_i \delta(v, f(x_i))$

Table 4: The distance-weighted k nearest neighbour algorithm.

Similarity is implemented as the Euclidean distance, so the algorithm classifies the input vector (which represents a cluster in this case) to its nearest neighbour (hence the name). The results will usually be better if multiple neighbours are considered in stead of just one. It is possible to take a vote among the neighbours to select the winning class, but the result will get more trustworthy if the vote is distance-weighted. This means that the distance of each neighbour involved will count in the ultimate decision of classification. The final algorithm that includes both these improvements is called the distance-weighted k nearest neighbour algorithm [19].

The distance-weighted k nearest neighbours algorithm belongs to the class of local weighted regression techniques. The target output is approximated based on the data available near the query point, and the contribution of each data point is weighted by the distance from the query point (hence the name *distance weighted regression*). The term regression is used because it is used in the domain of statistics in approximation of a certain real-valued function.

The simple nearest neighbour algorithm is very similar to the leader clustering algorithm, but in this case there is no feature that makes a new instance if the

similarity is not convincing. Both algorithms depend on the Euclidean distance to introduce similarity, however the nearest neighbour *classifies* data while the leader algorithm is restricted to *clustering*.

5.3. Radial Basis Function Networks

Radial Basis Function (RBF) networks are related to local weighted regression, as well as neural networks. The architecture is similar to that of a neural network with a hidden layer for which the combination function is based on the Euclidean distance between the input vector and the weight vector.

The hidden neurons produce an activation, which is determined by a Gaussian function with a certain mean and variance. Therefore the activation will only be significant if the input is close enough to the mean, otherwise it will be near zero. The neurons from the output layer combine the activations linearly.

RBF networks do not really have anything that is exactly the same as the bias term in a multi-layer perceptron. Some types of RBFs have a width associated with each hidden unit or with the entire hidden layer; instead of adding it in the combination function like a bias, the Euclidean distance is divided by the width. The output activation function in RBF networks is usually the identity. The identity output activation function is a computational convenience in training but it is possible and often desirable to use other output activation functions.

Although there are many types of radial basis functions, Gaussian RBF's seem to be the most popular by far in the neural network literature. There are two distinct types of Gaussian RBF architectures. The first type uses the exponent activation function, so the activation of the unit is a Gaussian as a function of the inputs. This kind of RBF is usually referred to as ordinary RBF or ORBF. The second type of Gaussian RBF architecture uses the softmax activation function, which means that the activations of all hidden units are normalized to sum one. This type of network is often called a normalized RBF or NRBF network. In this kind of architecture, the output units should not have a bias since the constant bias term would be linearly dependent on the constant sum of the hidden units.

The radial basis function network provides a global approximation to the target function, represented by a linear combination of many local kernel functions. The value for any given kernel function is non-negligible only when the input falls into

the region defined by its particular center and width. Thus, the network can be viewed as a smooth linear combination of multiple local approximations to the target. One main advantage to RBF networks is that they can be trained much more efficiently than the more common feedforward architectures trained with backpropagation. This follows from the fact that the input layer and the output layer of RBF networks are trained separately.

To use the radial basis function for classification, it is common to cluster the data first (as is done in this thesis), and use the resulting clusters as radial centers of the Gaussians.

5.4. Kohonen's LVQ

If the Self-Organizing Map is to be used as a classification algorithm, the problem becomes a decision process. Kohonen [15] modified the KSOM into a classification algorithm, and called it Learning Vector Quantization (LVQ). The classes are predefined and there is a set available of tagged data; every input vector is labelled with its correct class. It is very common to have several codebook vectors or prototypes per class.

The original Kohonen SOM is primarily intended to approximate input signal values by codebook vectors that are placed in the input space to obtain the best clustering results. This is obtained by minimizing the quantization error function. If the inputs are to be classified into a finite set of classes or categories, the codebook vectors are linked to a class and their identity within the classes is not important anymore. Only the decisions at the borders of classes will count in that case. The winning rule from the KSOM algorithm remains unmodified, but the update rule depends on whether the class of the winner is correct or not. The correction comes from a 'teacher', so this algorithm is a supervised one.

5.4.1. LVQ1

The first version of LVQ (LVQ1) [15] starts with a classification of some neurons, while others have no class attached to it. The class regions are defined with the nearest neighbour algorithm. The classification accuracy is enhanced if the codebook vectors are updated to the algorithm in Table 5. The idea behind the

algorithm is to pull codebook vectors away from the grey zones so that the class borders become more apparent.

\mathbf{m}_c is the codebook vector that is closest to the input \mathbf{x} , so this will define the classification of \mathbf{x} (see the nearest neighbour algorithm in section 5.2.).

Update the codebook vectors $\mathbf{m}_i = \mathbf{m}_i(t)$ as follows (where $\alpha(t)$ is the learning rate):

If \mathbf{x} is classified correctly:

$$\mathbf{m}_c(t+1) = \mathbf{m}_c(t) + \alpha(t) \cdot [\mathbf{x}(t) - \mathbf{m}_c(t)]$$

If the classification is incorrect:

$$\mathbf{m}_c(t+1) = \mathbf{m}_c(t) - \alpha(t) \cdot [\mathbf{x}(t) - \mathbf{m}_c(t)]$$

And $\mathbf{m}_i(t+1) = \mathbf{m}_i(t)$ for $i \neq c$

Table 5: The LVQ1 algorithm.

5.4.2. LVQ2

The second version of LVQ adds a symmetric window of nonzero width between two codebook vectors. Figure 21 illustrates the use of this window: the two circles depict two codebook vectors that belong to different classes and are nearest neighbours. The two are initially in a wrong position. The curves around them represent the according class distributions. The classification will be adjusted only if the input falls into the window and the classification was incorrect.

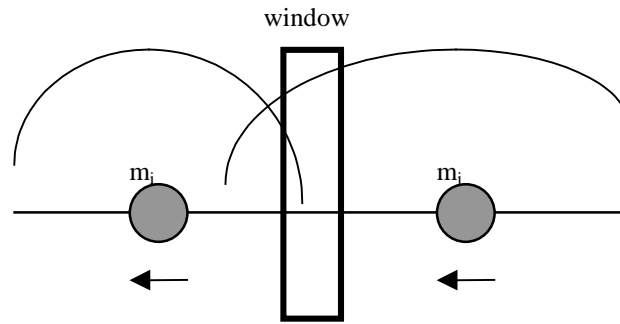


Figure 21: The window of the LVQ2 algorithm.

The optimal size of the window depends on the number of inputs that are presented to the network during the training phase. A narrow window guarantees the most accurate location of the border between classes. However, if the number of input samples is small and the window is very narrow, problems could occur in updating misclassifications. The input has to fall often enough in the window.

5.4.3. LVQ3

The final version of the algorithm provides a remedy for two problems that remain with the LVQ2 algorithm:

The update rules have the effect that the correction on the correct class is larger than that of the incorrect class, since the differences between the codebook vectors and the input are considered. This results in a decreasing distance between the two codebook vectors.

LVQ2 could lead to an asymptotic equilibrium of the wrong class that is not optimal. Corrections seem needed to ensure the continuing approximation of the class distributions.

The pseudo code of the LVQ3 algorithm is shown in Table 6. Kohonen found the optimal value of ϵ to depend on the size of the window, being small for narrow windows (between 0.1 and 0.5). The second and third version of LVQ change two codebook vectors at once, whereas in the first version only one was changed.

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) - \alpha(t) \cdot [\mathbf{x}(t) - \mathbf{m}_i(t)]$$

$$\mathbf{m}_j(t+1) = \mathbf{m}_j(t) + \alpha(t) \cdot [\mathbf{x}(t) - \mathbf{m}_j(t)]$$

Where \mathbf{m}_i and \mathbf{m}_j are the two closest codebook vectors to the input \mathbf{x} and its class should be that of \mathbf{m}_j and not that of \mathbf{m}_i . Furthermore, the input \mathbf{x} should fall in the window.

$$\mathbf{m}_k(t+1) = \mathbf{m}_k(t) + \varepsilon \cdot \alpha(t) \cdot [\mathbf{x}(t) - \mathbf{m}_k(t)] \text{ for } k \in \{i, j\}$$

Where the input and both \mathbf{m}_i and \mathbf{m}_j belong to the same class.

Table 6: The LVQ3 algorithm.

5.5. Minimizing the user intervention

Since the clustering is adaptive, it allows the further classification to be maintained and adjusted when necessary. This results in a minimal amount of user-intervention, since linking the clusters to context profiles is the only action a user has to be involved in. In contrast of the training-phase/test-phase method, where the recognition is executed with a stable but non-plastic system after it has been trained, the system stays plastic. There is in fact no distinction between a training- and test-phase of the system. This also means that the network trains itself for new contexts and the user can label the newly formed clusters afterwards. Thus, the system does not need to wait for the user to begin adapting itself to new contexts. The goal is to make the whole process as autonomous as possible, which will render it simultaneously more user-friendly.

Figure 22 shows the two places where the user has to intervene in the process of context recognition. The first one is the linking of clusters to context-identifiers that were initialized by the system. The second interaction is the labelling of these context identifiers with short descriptions.

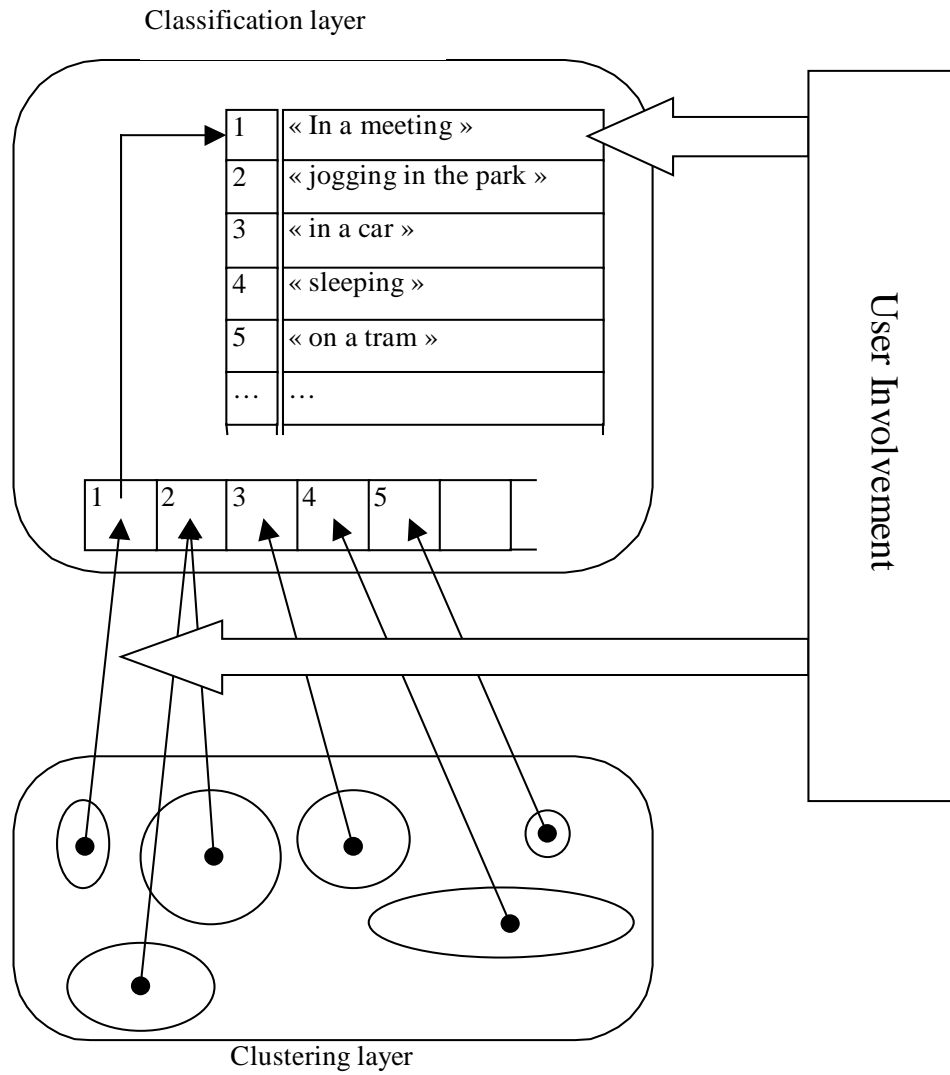


Figure 22: Schematics of the user involvement.

Chapter 6

The supervising layer

The system that has been discussed so far is able to associate a predefined context profile to a set of sensor values at a certain moment in time. In practice, there are no guaranties that the system that has been described up until now will always recognise the current context at exactly every moment in time. Another layer is needed to reach constant recognition of a context: the supervising layer. This top-down part of the system will also make user supervision possible and will improve the adaptive behaviour and recognition success of the learning system. Furthermore, it becomes possible to model sequences of contexts and thus a crude model of user behaviour.

6.1. A probabilistic finite state machine

The probabilistic finite state machine contains two elements: states and actions. The difference with a normal finite state machine is that the actions are labelled with the probability that they could be chosen. Every user-defined context profile will be represented by a state, while transitions of states will be indicated by an action that goes from one context to another. Every transition has a label attached to it that contains an indication of how probable it is that a transition between the two contexts occurs. This way, transitions of states can be checked on probability and it is possible to keep track of the probability of entire sequences of states. This information can be used as a sort of emergency backup if the recognition system has substantial problems with the classification. The probabilistic finite state machine can pick out the most probable state, based on the previous states.

The probabilistic finite state automaton is very general since it also keeps track of previous states. This is not really necessary and it is sufficient to look at the previous state for the choice of actions in this application. Reducing the probabilistic finite state machine in this fashion leads to a Markov chain model.

6.2. Markov chains

A Markov chain is a sequence of events or states where the chance of each occurring is solely dependent on which state preceded it. In a probabilistic finite state automaton multiple previous states count in the decision process. In the diagram below (Figure 23) a Markov chain is described which consists solely of the words in the sequence ‘the cat sat on the mat’. Each word represents a state that the system can adopt, and the arrows joining them show the probabilities of shifting from one state to another one.

Many names exist in literature to label this graph, although Markov chain and Markov model are most used. Among their synonyms are first-order Markov chain or first-order Markov model (in contrast, a higher order Markov model would be closer to a probabilistic finite state machine) which are also used frequently.

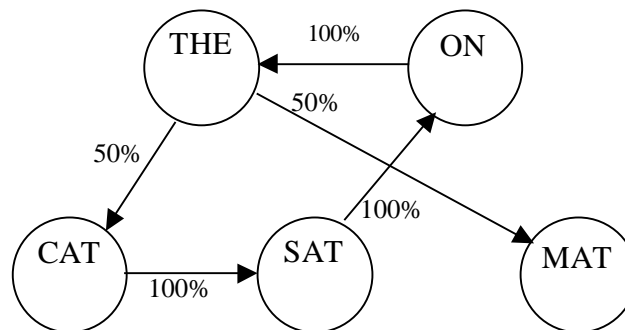


Figure 23: Example diagram of the Markov model.

For example, as far as the laws of grammar are concerned, the only word in this selection that can follow ‘on’ is ‘the’. To put it in a more general way, the probability of ‘the’ being the next word in the sequence is 100 percent, while the probabilities of ‘cat’, ‘sat’ or ‘mat’ are zero (these links are not shown in the graph). By contrast, ‘the’ is just as likely to be followed by ‘mat’ or ‘cat’, so these paths are given an equal 50 percent chance. In practice, a Markov model would be more complex, it would involve more states and their probabilities would not be nearly so evenly apportioned.

Imagine that the Self-Organizing Map (or a similar clustering algorithm) has correctly identified the first few context profiles, but there remains some doubt on what the next will be. A Markov model could quickly decide between the two. If the previous context was for instance ‘in a train’, the Markov chain would know that it

would be more likely that the next context would be ‘in a train station’. It would be highly unlikely that the next context would be ‘in a bedroom’ for example.

The Markov chain is often implemented as a matrix, where each column and row represent the probabilities to go from one state to the other.

Using the Markov model to add learning of long-term variations to the system is widely used in various recognition systems. The most important advantage of the Markov chain is that it is able to deal with variable length sequences. Hand sign recognition [28] and lipreading [12] using a Markov model were implemented with good results.

6.3. Introducing adaptivity

The user can supply the data for this supervising layer, both for the context descriptions and for the probabilities of the transitions. Nevertheless, it is also possible –and perhaps preferable- to make the Markov Model adaptive towards the probabilities. If the system keeps track of the number of transitions between two certain contexts, it will become possible to update automatically the probabilities in the model. It is also possible to combine user interaction and these self-adapting probabilities.

In Table 7, the simple algorithm is summarised to update the probabilities in an adaptive way. Every directed link from one context profile to another has two numbers attached to it: a variable that counts how many times the transition has been made, and the probability of that transition. If the counting variable gets too high, all counting variables can be divided by a constant larger than one to escape from getting overflow errors.

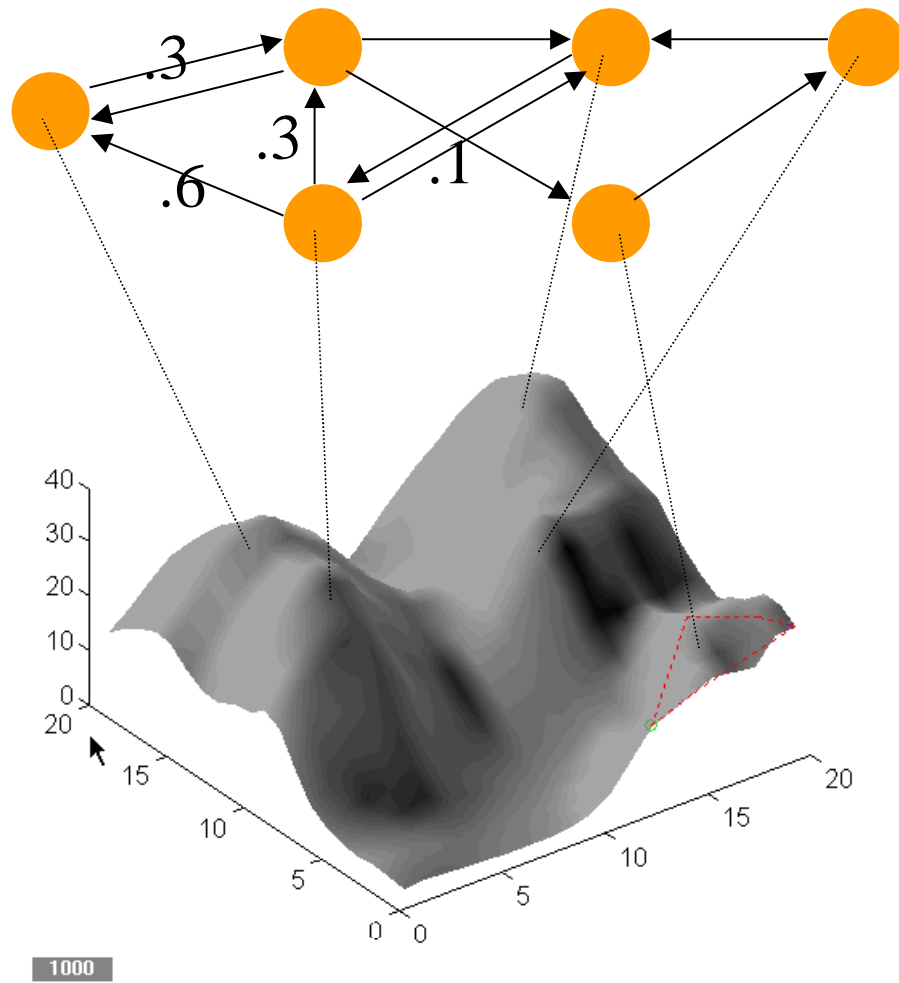


Figure 24: Example diagram of the Markov model of clusters.

Every time a transition has been made to another context:

Add one to the counting variable connected to the arrow going from the previous context to the current one.

For every state in the Markov Chain (or every context profile):

Transform all counting variables into percentages to obtain the probabilities.

Table 7: The steps to update the probabilities in the Markov chain.

The probabilistic finite state approach introduces a notion of habit for the behaviour profile of a specific user. Storing the context profiles as states and the

transitions as edges makes it possible to learn a model of the habits of the user with relation to the frequency and the sequence of the contexts.

6.4. Clusters

Another possibility is to link the clusters from the clustering layer directly to the states in the Markov chain. This approach is feasible if the number of clusters is not too large to avoid a huge (and consequently slow) Markov chain.

Again, the feasibility of implementing this method depends on the size of the competitive layer, or more generally the number of clusters from the clustering layer. It was already assumed that the clustering layer was composed out of multiple Self-Organizing Maps acting in parallel, so this method will not be further investigated.

6.5. Experiments and results

The Markov chain intends to refine the recognition in two ways:

- by providing a fallback mechanism when the classification layer has not enough information to successfully make a recognition.
- by upkeeping a model of the user's behaviour with relation to both the frequency and the order of contexts.

Experiments that test the ability of the system on both these improvements demonstrate that they really facilitate the recognition. As mentioned earlier, evaluating the recognition success rate is fairly simple: after a while, the system should always be able to recognize the correct context. If this is not the case, inadequate sensors or cues are most likely to be the reason for failure. Thus, the speed of recognition is the factor that has to be maximised.

The experiment described at the end of chapter 4 has already used a Markov chain to show the ultimate recognition results while using different clustering methods. The probabilities of the Markov chain were formed during a training phase in which the contexts were introduced in the sequence 1-2-3-4-5. This resulted in a correct and continuous overall recognition of the contexts with a delay of a few seconds.

Although the resulting chain is a very simple one, it illustrates how the Markov chain could contribute to a better recognition. If the model would be more complex, transitions might get tougher to learn.

Chapter 7

Improving the performance

This system has a lot of parameters and algorithms that can be modified, replaced by another one, or even left out, and that change will usually have an impact on the performance of the system. Apart from these algorithm-related modifications, the choice of input sensors, sensitivity of the sensors and the frequency of polling are also crucial to the final behaviour of the system. These hardware-related matters are not within the scope of this thesis and will therefore not be discussed here.

7.1. The architecture

The number and nature of the sensors has a big impact on the recognition system. If there are only a few sensors available, it is not really necessary to include an intermediate layer that uses multiple clustering algorithms in a hierarchical way. The low input dimensionality from a few sensors does not cause the algorithms to behave slowly, so one Self-Organizing Map is sufficient in this case. If, on the other hand, the number of sensors is high, partitioning of the input space is feasible, as indicated in the experiments.

Cues are very important in the design of the adaptive system, since they are in fact calculations that do not need to be determined anymore by generalisation in the neural network. They can take a lot of work out of the hands of the clustering layer by using fast real-time pre-processing that otherwise would need to be done by (slower) generalising by the neural network. The outcome is a faster and more precise system.

The probabilistic finite state module has been simplified in the previous chapter, but some applications might require a more complex implementation. If the current context depends on more than the most recent past context, a higher order Markov model could be used. This would result in heavier processing requirements, however.

7.2. Parameters of the SOM and RSOM

The architecture of the SOM is very important to the quality of the clustering. The value of the learning rate, the size (and dimension) of the output layer and the neighbourhood radius have a substantial impact on the final result. Although there is no mathematical proof of this, it is generally accepted that a two-dimensional output layer performs very well and it is also very easy to implement. The optimal size of the layer is not so easy to obtain, since it depends on the nature of the output space, the number of clusters that can appear and the dimension of the input space.

The learning rate was in most experiments chosen around 0.05 to avoid unlearning, while the neighbourhood radius was approximately three (with the Gauss function as neighbourhood function).

The initial status of the competitive layer and its set of weights is very important for the quality of clustering. Instead of using random weights, it is also possible (and better) to initialize the weights with a small training phase, using a large learning rate and neighbourhood radius and multiple different input vectors. It was experimentally verified that the latter results in more precise and faster clustering and learning.

7.3. What (combination of) sensors will perform best?

In order to know what contribution a certain sensor makes to the recognition of a certain context, it might be necessary to make a preliminary system that is able to detect that sensor's performance – not just individually, but also in combination with other sensors. This will make it easier to evaluate the performance of a sensor and the contribution it makes to the recognition system.

The evaluation system has to measure the performance of a set of sensors. This requires a solution to another problem: what sensors should be grouped together to obtain an optimal result? As seen in chapter 4, it is crucial to know this when splitting up the total set of sensors to avoid throwing away relevant combinations. It is however next to impossible to try all possibilities and compare the results, even if the number of sensors is relatively low.

The activation surface described in chapter 3 gives a good informal indication on how the choice of sensor-groups affects the behaviour of the system, and comes closest to a feasible evaluation method. It is far from perfect and not formal. In most

applications, however, the grouping of sensors can be done in an intuitive way to some extent.

Chapter 8

Future work

This chapter contains a list of some of the work that still could (or need) to be done to enhance the performance of the system in terms of recognition precision and speed. Expanding the system for more complex context models like hierarchical or overlapping structures is discussed, followed by rule extraction possibilities. Finally, a method is proposed to make the clustering more reliable.

8.1. Implementation of user supervision and interaction

Although the system was designed to need as little user interventions as possible, the potentiality to train the system on-line has not been implemented in the experiments. How the user should on-line interact with the system is a matter that has not been solved yet, however a number of possibilities exists.

8.1.1. No user feedback

It is possible to pre-train the system for contexts that are general and can be distinguished in every case. Contexts like “in a car” and “jogging” have similar effects on the sensor-readings all over the world, in any situation. One car can be a bit noisier than the other, or one person can move differently during jogging than another person, but in general these contexts have sensor-outputs or cues that are ‘fixed’. The recognition of the jogging context for instance will probably rely on cues from the acceleration sensors.

Translate this to the cluster-terms from chapter 3 and the outcome will be like this: the training phase can be done by making clusters in the clustering layer from prototype contexts that are as general as possible. An average man or woman that jogs for a while would be the choice in the jogging context. Clusters will emerge during this training on the Kohonen map that will represent the ‘average jogging context’. When, after the training phase, the system is then put to use, the clustering layer will automatically adapt these clusters to the specific contexts of the user. To say it in a more visual way: the hills on the activation surface will reshape or resize themselves from general prototype clusters towards the user’s own tailor-made

clusters. The error rate of the recognition process might be high in the beginning of the system's usage, but while time goes by the system will improve itself by enhancing the general contexts to specific contexts.

8.1.2. Some user feedback

If the application for the system requires that the user should be able to add context profiles, the previous approach will not be sufficient. Since the clustering layer is unsupervised, the intervention of the user will probably need to have an impact on the parameters of the classification- or the Markov chain layer. There are two types of data that the user must supply to the system:

- Context profiles: typical data and the corresponding label:

The data will be obtained by the device and clustered by the first layer, after which these clusters will be given the label that the user supplied. More concrete, the user will input a context profile name and press some kind of training button that indicates the system should label the incoming data with the context profile that was just supplied.

- Probabilities in the Markov chain:

This interaction with the system is not as vital for the recognition as the previous one, but can be useful to add transparency in the workings of the supervising layer. The user can be allowed to edit the probabilities directly or partially to prevent the input of contradictory information.

8.2. *Non-exclusive contexts*

Another domain that was not explored in this thesis is the implementation of non-exclusive context profiles, i.e. more than one context that can be valid for the same raw sensor data. The easiest way to implement this feature is probably by building multiple Markov chains that run in parallel on the same clustering layer.

The clustering method remains unchanged in this architecture, but in this approach it is possible that clusters get associated to more than one context in contrast with using only exclusive contexts.

Another, more complex, option may be a hierarchical structure of contexts, as the one depicted in Figure 25. It is possible to partition the hierarchy in several

layers, where each layer can have a different level of abstraction or can belong to a different aspect of context. The “Indoors” and “Outdoors” contexts are for instance more general than the “Kitchen”, “Office” and “In park” contexts. These contexts are more related to location, while the “Cooking”, “Eating”, “Meeting” and “Jogging” contexts are more activity related.

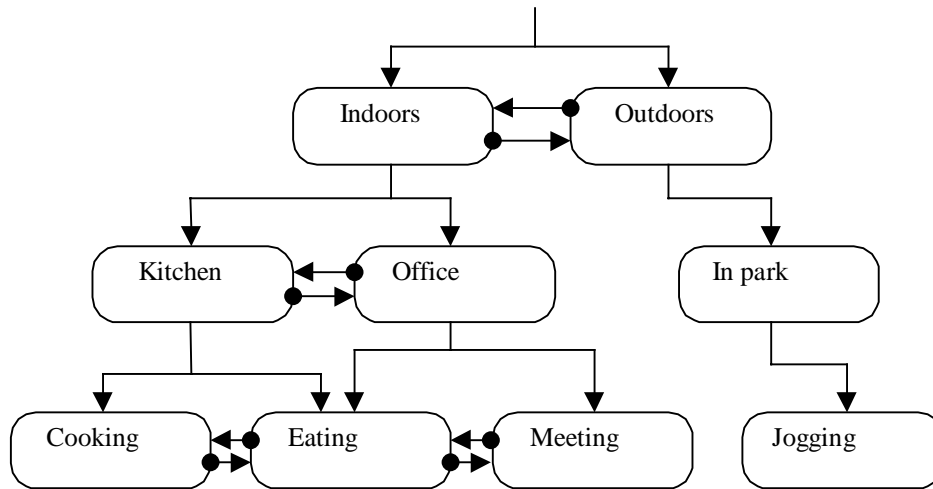


Figure 25: Example diagram of a hierarchical architecture.

Another issue in the case of a hierarchical structure is whether the activity should be placed on the top of the location or vice versa. In the figure, the choice of location above activity was taken, but this is not actually necessary. This method is probably not sufficient, since it is hard to include different levels of abstraction for both activity and location. The activity could for instance be divided into two levels: one higher level with “Moving” and “Not moving” as context profiles, and a lower level with contexts like the ones illustrated in Figure 25.

It is also possible to combine the two approaches above: two parallel hierarchical graphs, one with activity-related context profiles, and one with location-related contexts. This might solve the problem with implementing different levels of location or activity.

8.3. Rules Extraction

After the clustering and classification of the cues and sensor data, it is possible to do rule extraction out of the learned model. This would be convenient if the user wants to have not only transparency in the outcome of the system, but also clarity in

the mechanics of the learning system itself. The latter cannot be achieved in the current architecture, unless rule extraction is introduced.

An example would be for instance (verified by experimentation):

Hand(t) :- standard_derivation(accelX,t) > Dx,
 standard_derivation(accelY,t) > Dy,
 average(light,t) > L.
 Table(t) :- abs(average(accelX,t)-Xnormal) < D,
 abs(average(accelY,t)-Ynormal) < D,
 quartile(accelX,t) < Q, quartile(accelY,t) < Q,
 average(light,t) > L.
 Suitcase(t):- average(light,t) < L.

where hand, table and suitcase are descriptors of contexts. Hand stands for the device being in the user's hand (the device is slightly moving in x and y direction and it is not totally dark). Table represents the device resting on a table (no movement, not dark). Suitcase defines the context of the device being in a suitcase (totally dark). These simplified recognition rules are fairly easy to read and to understand.

8.4. Top-down stabilisation mechanism

The Adaptive Resonance Theory mentioned in chapter 3 contains a top-down mechanism that makes it possible to obtain a stable and plastic system. Novel inputs directly access a category if they share invariant properties with the codebook vector of the cluster. The system becomes highly plastic when fresh inputs appear to enable fast training. After a while this plasticity diminishes to keep the system stable.

New input patterns in the case of this thesis can be recognised by the Markov chain if for instance the user adds a new context profile for training. Increasing the learning rate in that case enables the clustering neural network to enhance its training and learn to cluster similar patterns much faster.

Another possibility could include the use of parameters attached to every context-state in the Markov chain that indicate how long the user was present in that context. The learning rate and neighbourhood parameters could therefore be adjusted so that the clustering of signals from every context could be equal. This offers also a solution to the training problems that occur if the user visits some contexts significantly more than others.

The result is that the whole system becomes even more adaptive if this feedback from the supervising layer is implemented.

Chapter 9

Conclusions

Merging a bottom-up connectionist approach and a top-down symbolic method results in an adaptive on-line system that is able to recognise and distinguish contexts. Furthermore, it does not need a bulk of user-involvement to reach its goal. Despite the fact that the complexity of the experiments were somehow limited by the hardware (as they will be in the future, since there is theoretically no real limit in the number and variety of sensors), very promising results emerged from this thesis.

9.1. The architecture

In this thesis, a hybrid architecture is proposed to map raw sensor data to a high-level context profile, created by the user. It combines an unsupervised neural network and a probabilistic finite state machine to reach a maximum level of reliability and adaptivity and a minimal amount of user-involvement. The architecture is able to adjust itself during on-line recognition.

Figure 26 illustrates how the final architecture would look like. The clustering layer (described in chapter 3) is implemented by three SOMs in a hierarchy (explained in chapter 4). The outputs of the SOMs are clustered again by the leader clustering algorithm, which delivers the final output of the clustering layer. The classification is done by simply labelling every cluster to a context profile (this and other methods can be found in chapter 5). The classes are also connected to a state in a Markov chain, which provides the implementation of the supervising layer, as described in chapter 6.

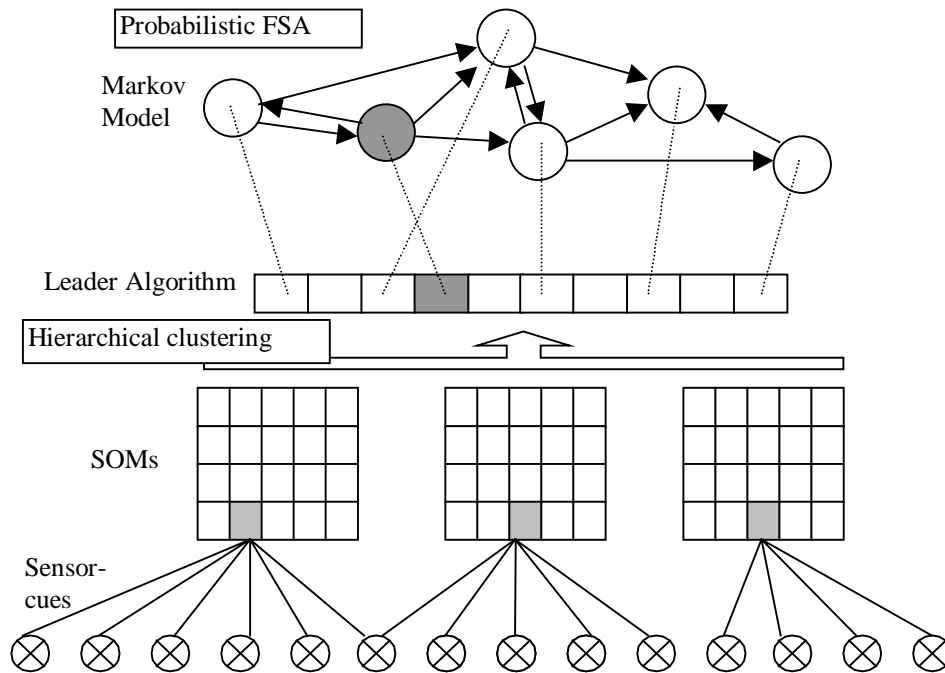


Figure 26: Example diagram of the final architecture.

The Self-Organizing Map has proven to be a good clustering method, which is able to remain adaptive and does not slow the learning process down. Although the topology-preserving ability of the Kohonen neural network is not always necessary for the clustering, it was implemented in every experiment in this thesis for visualization purposes. Both the plasticity-stability dilemma and the curse of dimensionality can be avoided if the dimension of the input space is small enough. If the number of sensors or the number of cues grows too large, however, there still is the possibility to combine multiple algorithms in a hierarchy. Experiments show that this approach is feasible. If this hierarchical structure is used with a second-layer clustering algorithm (like the leader clustering algorithm, as shown in the picture), the topology preserving property from the Kohonen network will be necessary.

The behaviour over time is processed on two levels: a sub-symbolic level (the RSOM) and a symbolic level (the Markov chain). The sub-symbolic level takes care of short-term patterns that appear in a few seconds at most, while the symbolic level does longer term time-pattern processing.

The probabilistic finite state automaton, in the simplified form of a Markov Chain, adds a long-term predictive component that has three major benefits:

1. It enhances the recognition precision by introducing probabilities.

2. It implements some recognition of user behaviour (how frequently and in what order contexts appear for a certain user).
3. It gives the user a better overview and the possibility to inspect the behaviour of the system.

The final system is adaptive in many ways: the adaptive clustering layer ensures that the system will adjust itself to the specific contexts in which the user has been. The probabilistic finite state machine layer (implemented as a Markov chain) provides the whole with a form of behaviour recognition. The first can be interpreted as a short-term pattern recognition of specific contexts, while the latter can be seen as long-term pattern recognition of general user behaviour with relation to contexts.

9.2. Accomplishments

Although all experiments and the final system were implemented in Matlab – which does not make it easy to link the programs to others or to implement them into hardware -, they were valuable for the TEA project. The feasibility of the hybrid architecture for an on-line adaptive context aware system was indicated in the experiments and the system's performance and on-line ability was demonstrated.

The early visualisation and clustering methods were presented and included in a demo at the Starlab NV/SA inauguration days. The architecture of the hybrid system was explained and illustrated at the TEA demonstration in Bologna, Italy.

Another concrete result was the publication of the initial thesis results in a paper by Schmidt, Asante Aidoo, Takaluoma, Tuomela, Van de Velde and the author [27] which describes the evaluation and accomplishments of the TEA project after its first year. It was submitted for the 1999 international symposium on Handheld and Ubiquitous Computing in Karlsruhe, Germany.

Bibliography

1. G. D. Abowd. Software Engineering Issues for Ubiquitous Computing. *In the Proceedings of ICSE'99*. 1999.
2. H.W.P. Beadle, B. Harper, G.Q. Maguire Jr. and J. Judge. Location Aware Mobile Computing. In *Proc. IEEE/IEE International Conference on Telecommunications (ICT'97)*, pages 1319-1324, Melbourne. 1997.
3. H.W.P. Beadle, G.Q. Maguire Jr. and M.T. Smith. Environment Aware Computing and Communication Systems. *In Proc. International Symposium on Wearable Computers*. Cambridge, Massachusetts. 1997.
4. H.W.P. Beadle, G.Q. Maguire Jr. and M.T. Smith. Using Location and Environment Awareness in Mobile Communications. *In Proc. IEEE/IEEE ICICS'97*, pp. 1781-1785, Singapore. 1997.
5. R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press. 1961.
6. G. A. Carpenter and S. Grossberg. A massively parallel architecture for a Self-Organizing Neural Pattern Recognition Machine. *In Computer Vision, Graphics, and Image Processing 37*, pages 54-115. Academic Press, Inc. 1987.
7. C. Chen. Sensor fusion. 1999.
8. J. Eriksson and N. Finne. The Context Sensitive Feeling Injector, or Safe Tourist. Teleinformatics course, SICS Sweden. 1998.
9. Esprit Project 26900. Technology for enabling Awareness (TEA). www.omega.it/tea/. 1998.
10. N. R. Euliano and J.C. Principe. Spatio-Temporal Self-Organizing Feature Maps. *In Proc. of the ICNN*, pages 1900-1905, Washington. 1996.
11. B. Fritzke. *Some Competitive Learning Methods*. Artificial Intelligence Institute, Dresden University of Technology. 1997.
12. A.J. Goldschen, O.N. Garcia and E.D. Petajan. Continuous automatic speech recognition by lipreading. In *Motion-Based recognition*, M. Shah and R. Jain (eds.), pages 227-243. Kluwer Academic Publishers, the Netherlands. 1997.
13. J. Hertz, A. Krogh and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison Wesley. 1991.
14. T. Honkela, V.Pulkki and T. Kohonen. Contextual Relations of Words in Grimm Tales, Analysed by Self-Organizing Map. In *Proc. of ICANN-95*, Eds. F. Fogelman-Soulie and P. Gallinari, EC2 et Cie, pages 3-7, Paris. 1995.
15. T. Kohonen. The Self-Organising Map. *In Proc. of the IEEE*, vol. 78, no. 9, pages 1464-1480. 1990.
16. T. Koskela, M. Varsta, J. Heikkonen and K. Kaski. Time series prediction using Recurrent SOM with local linear models.
17. C. Lau. *Neural Networks: Theoretical Foundations and Analysis*, IEEE press. 1992.
18. R. P. Lippmann. An introduction to Computing with Neural Nets. *In IEEE Acoustics, Speech, and Signal Processing Magazine*, April 1987, pages 4-22. 1987.

19. T. M. Mitchell. *Machine Learning*. McGraw-Hill. 1997.
20. D. A. Norman. Why Interfaces Don't Work. *The Art of Human-Computer Interface Design*. Brenda Laurel (editor). Addison-Wesley. 1992.
21. M. T. Rosenstein and P. R. Cohen. Symbol Grounding with delay coordinates. In *Working Notes of the AAAI Workshop on The Grounding of Word Meaning*. 1998.
22. M. T. Rosenstein and P. R. Cohen. Concepts from Time Series. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 739-745, AAAI Press. 1998.
23. M. T. Rosenstein and P. R. Cohen. Continuous Categories for a Mobile Robot. In *Proc. of the AAAI '99*. 1999
24. S. Sagaert. The Stable Growing Neural Gas. Masters Thesis at the Vrije Universiteit van Brussel. 1996.
25. N. Sawhney. Situational Awareness from Environmental Sounds, final report for Modeling Adaptive Behaviour (MAS 738). 1997.
26. A. Schmidt, M. Beigl, H. W. Gellersen, There is more to context than location. In *Proc. of the Intl. Workshop on Interactive Applications of Mobile Computing (IMC98)*, Rostock, Germany, November 1998.
27. A. Schmidt, K. Asante Aidoo, A. Takaluoma, U. Tuomela, K. Van Laerhoven and Walter Van de Velde. Advanced Interaction in Context. Submitted for 1999 International symposium on Handheld and ubiquitous computing in Karlsruhe. 1999.
28. T. Startner and A. Pentland. Real-time American sign language recognition from video using hidden Markov Models. In *Motion-Based recognition*, M. Shah and R. Jain (eds.), pages 227-243. Kluwer Academic Publishers, the Netherlands. 1997.
29. M. Varsta, J. Heikkonen and J. Del Ruiz Millán. Context Learning with the Self-Organizing Map. In *Proc. on Self-Organizing Maps WSOM'97*, pages 197-202, Helsinki University of Technology. 1997.
30. M. Weiser. Some Computer Science Problems in Ubiquitous Computing. In *Communications of the ACM*, July 1993.