THE OPTIMIZATION OF GESTURE RECOGNITION TECHNIQUES FOR RESOURCE-CONSTRAINED DEVICES

by

Gerrit Niezen

Submitted in partial fulfillment of the requirements for the degree

Master of Engineering (Computer Engineering)

in the

Faculty of Engineering, the Built Environment and Information Technology

UNIVERSITY OF PRETORIA

April 2008

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Summary

THE OPTIMIZATION OF GESTURE RECOGNITION TECHNIQUES FOR RESOURCE-CONSTRAINED DEVICES

by

Gerrit Niezen

Supervisor:	Prof. G.P. Hancke
Department:	Electrical, Electronic and Computer Engineering
	UNIVERSITY OF PRETORIA
Degree:	M.Eng. Computer Engineering

Gesture recognition is becoming increasingly popular as an input mechanism for human-computer interfaces. The availability of MEMS (Micro-Electromechanical System) 3-axis linear accelerometers allows for the design of an inexpensive mobile gesture recognition system. Wearable inertial sensors are a low-cost, low-power solution to recognize gestures and, more generally, track the movements of a person.

Gesture recognition algorithms have traditionally only been implemented in cases where ample system resources are available, i.e. on desktop computers with fast processors and large amounts of memory. In the cases where a gesture recognition algorithm has been implemented on a resource-constrained device, only the simplest algorithms were implemented to recognize only a small set of gestures.

Current gesture recognition technology can be improved by making algorithms faster, more robust, and more accurate. The most dramatic results in optimization are obtained by completely changing an algorithm to decrease the number of computations. Algorithms can also be optimized by profiling or timing the different sections of the algorithm to identify problem areas.

Gestures have two aspects of signal characteristics that make them difficult to recognize: segmentation ambiguity and spatio-temporal variability. Segmentation ambiguity refers to not knowing the gesture boundaries, and therefore reference patterns have to be matched with all possible segments of input signals. Spatio-temporal variability refers to the fact that each repetition of the same gesture varies dynamically in shape and duration, even for the same gesturer.

The objective of this study was to evaluate the various gesture recognition algorithms currently in use, after which the most suitable algorithm was optimized in order to implement it on a mobile device. Gesture recognition techniques studied include hidden Markov models, artificial neural networks and dynamic time warping. A dataset for evaluating the gesture recognition algorithms was gathered using a mobile device's embedded accelerometer. The algorithms were evaluated based on computational efficiency, recognition accuracy and storage efficiency. The optimized algorithm was implemented in a user application on the mobile device to test the empirical validity of the study.

Keywords: gesture recognition, linear accelerometer, neural networks, hidden Markov models, dynamic time warping, mobile devices, optimization, human-computer interfaces, wearable computing

Opsomming

DIE OPTIMALISERING VAN GEBAARHERKENNINGSTEGNIEKE VIR HULPBRON-BEPERKTE TOESTELLE

deur

Gerrit Niezen

Studieleier:	Prof. G.P. Hancke
Departement:	Elektriese, Elektroniese en Rekenaaringenieurswese
	UNIVERSITEIT VAN PRETORIA
Degree:	M.Ing. Rekenaaringenieurswese

Gebaarherkenning is besig om meer en meer populêr te word as ’n inset-meganisme vir mens-rekenaar koppelvlakke. Die beskikbaarheid van MEMS (Mikro-elektromeganiese Stelsel) 3-as lineêre versnellingsmeters maak dit moontlik om ’n goedkoop draagbare gebaarherkenningstelsel te ontwerp. Draagbare inersie-sensors is ’n goedkoop, lae kragverbruiksoplossing om gebare te herken en in die algemeen, die bewegings van ’n persoon te kan bespeur.

Gebaarherkenningsalgoritmes is tradisioneel net geïmplementeer in gevalle waar volop stelselhulpbronne beskikbaar is, d.w.s. op tafelrekenaars met vinnige verwerkers en groot hoeveelhede geheue. In die gevalle waar ’n gebaarherkenningstelsel op ’n hulpbron-beperkte toestel geïmplementeer is, is net die eenvoudigste algoritmes gebruik om net ’n klein hoeveelheid gebare te herken.

Huidige gebaarherkenningstegnologie kan verbeter word deur die algoritmes vinniger, meer robuus en meer akkuraat te maak. Die mees dramatiese resultate in optimalisering kan verkry word deur die algoritme heeltemal te verander om die hoeveelheid bewerkings te verminder. Algoritmes kan ook geoptimaliseer word deur die profielsamestelling of die tydmeting van die verskillende onderafdelings van die algoritme te bepaal, om moontlike probleemareas te identifiseer.

Gebare beskik oor twee aspekte van sein-karakteristieke wat hulle moeilik maak om te kan herken: segmentasie-dubbelsinnigheid en tydruimtelike veranderlikheid. Segmentasie-dubbelsinnigheid verwys daarna om nie te weet waar die gebaar se grense is nie en daarom moet verwysingspatrone met elke moontlike segment van die insetseine gepas word. Tydruimtelike veranderlikheid verwys na die feit dat elke herhaling van dieselfde gebaar dinamies verskil in vorm en duur, selfs as dit dieselfde persoon is wat die gebaar uitvoer.

Die doel van hierdie studie was om die verskillende gebaarherkenningsalgoritmes wat huidiglik in gebruik is te evalueer. Daarna is die mees geskikte algoritme geoptimaliseer om dit op 'n draagbare toestel te implementeer. Gebaarherkenningstegnieke wat bestudeer is, sluit verskuilde Markov-modelle, kunsmatige neurale netwerke en dinamiese tydsverbuiging in. 'n Datastel om die gebaarherkenningsalgoritmes te evalueer is versamel deur van 'n draagbare toestel se ingebedde versnellingsmeter gebruik te maak. Die algoritmes is geëvalueer op grond van hulle berekeningsdoeltreffendheid, herkenningsdoeltreffendheid asook bergingsdoeltreffendheid. Die geoptimaliseerde algoritme is in 'n gebruikerstoepassing geïmplementeer op die draagbare toestel om die empiriese geldigheid van die studie te toets.

Sleutelwoorde: gebaarherkenning, lineêre versnellingsmeters, neurale netwerke, verskuilde Markov-modelle, dinamiese tydsverbuiging, draagbare toestelle, optimalisering, mens-rekenaar koppelvlak, draagbare rekenaarstelsels

Acknowledgements

I would like to thank Professor G.P. Hancke, who kept me motivated to complete my studies and allowed me the opportunity to study under him.

A special thanks to my parents who supported me throughout my studies and provided the necessary encouragement. I would also like to thank my girlfriend, Lize, who had to cope with me during the more difficult times. I also thank all my friends, including my brother Riaan, who provided support and guidance when needed.

I would like to thank Tracy Westeyn at the Georgia University of Technology, who helped me to master the gesture recognition toolkit they developed.

I would like to give my deepest thanks to God - I would not be able to do anything without Him.

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Contents

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

List of Abbreviations

ANN	Artificial Neural Network
API	Application Programming Interface
AR	Auto-Regressive
ASCII	American Standard Code for Information Interchange
BN	Bayesian Network
CAD	Computer Aided Design
CTRNN	Continuous Time Recurrent Neural Network
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
DCE	Dead Code Elimination
DTW	Dynamic Time Warping
EM	Expectation-Maximization
FANN	Fast Artificial Neural Network
FFT	Fast Fourier Transform
FP	Feature Pack
FPU	Floating Point Unit
FSM	Finite State Machine
FWL	Fractional Word-Length
GCCE	GNU Compiler Collection for Embedded
GT2K	Georgia Tech Gesture recognition Toolkit
HCI	Human Computer Interface
HMM	Hidden Markov Model
HPF	High Pass Filter
HTK	Hidden Markov Model Toolkit
IDE	Integrated Development Environment
IIR	Infinite Impulse Response
IWL	Integer Word-Length

LPF	Low Pass Filter
MEMS	Micro-Electromechanical Systems
MIPS	Million Instructions Per Second
PCA	Principal Component Analysis
PDA	Personal Digital Assistant
POSIX	Portable Operating System Interface
RAM	Random Access Memory
RBF	Radial Basis Function
RFID	Radio Frequency Identification
RNN	Recurrent Neural Network
RPROP	Resilient Backpropagation
SBS	Sequential Backward Selection
SDK	Software Development Kit
SFS	Sequential Forward Selection
SOM	Self-Organizing Map
SPG	Self-Propelled Gun
SVM	Support Vector Machine
VQ	Vector Quantization

Chapter 1

Introduction

People are highly skilled in using gestures to communicate, yet few applications let them use gestures to control objects in the real world [1]. Using free-hand gestures allows the user to interact directly with an object in a natural way that is easy to learn. It allows for higher power of expression, with a single gesture being used to specify both a command and its parameters. In [2] it is argued that the most interesting devices will be those that integrate enough processing power to perform the software functions of gesture recognition and matching on the device itself.

While some research projects [3, 4, 5] do collect gesture data wirelessly using small sensor nodes, gesture recognition is still performed on a desktop computer. Gesture recognition algorithms have large processing and memory requirements that can usually not be performed on a resource-constrained device such as a microcontroller.

The most popular technique for recognizing gestures is by using visual gesture data from a camera-based sensor. This method is fairly precise, but also the most demanding in terms of setting up the infrastructure, maintaining the hardware and algorithm complexity [19]. This has consequences for the applicability of such systems: motion capture and tracking platforms are rarely used beyond CAD animation or medical purposes. Image-based gesture recognition techniques have resource demands that can make gesture recognition particularly difficult on PDAs and other resource-constrained devices [6]. The requirements for pre-positioned cameras and good lighting conditions make this approach unsuitable for mobile systems [4].

Other options include magnetometers that can be used to determine the orientation of a device, but they are prone to noise generated by electrical equipment such as televisions, speakers and mobile phones. Gyroscopes, or angular rate sensors, can prove costly when implemented on the three orthogonal axes, i.e. one sensor per axis.

The availability of MEMS (Micro-Electromechanical System) 3-axis linear accelerometers allows for the design of an inexpensive mobile gesture recognition system. A single integrated tri-axial accelerometer provides better accuracy than traditional orthogonally mounted accelerometers, and reduces re-calibration frequency [5]. Wearable inertial sensors are a low-cost, low-power solution to track gestures and, more generally, movements of a person.

By utilizing accelerometer-based gesture recognition techniques and optimizing the gesture recognition algorithms for resource-constrained devices, truly mobile applications in the field of Human-Computer Interaction (HCI) can be created by performing gesture recognition on-board a mobile device. Users will be able to manipulate objects on the mobile device directly using hand gestures, or use gestures to issue specific commands [1].

Sensor-based techniques have the advantage that computationally intensive calculations are not required for accurate movement information as measurements are directly provided by the sensors. Many researchers working with image-based sensors argue that cameras are unobtrusive compared to the often cumbersome sensor devices. Sensor-based techniques, however, have the advantage in that they can be used in much less constrained conditions and are not reliant on lighting conditions or camera calibration [27].

1.1 Overview of current literature

Gesture recognition techniques can be defined into two classes: discrete gesture recognition and continuous gesture recognition. Discrete gesture recognition is performed one gesture at a time, while continuous gesture recognition is performed on a sequence of gestures within a contiguous block of data [3]. Discrete gesture recognition uses an explicit command from the user to indicate the start and stop of the gesture, e.g. with a button. With continuous gesture recognition, the recognition is performed on a continuous flow of gestural input data [7].

Classical Bayes decision theory links a recognition task to the problem of distribution estimation. For pattern recognition, the approach is to do statistical pattern classification based on a discriminant function. For a given family of discriminant functions, optimal classifier/recognizer design involves finding a set of parameters which minimize the empirical pattern recognition error rate [8].

A classic gesture recognition system matches a sequence of hand positions over time to a number of prototype gesture sequences, each of which are learned from a set of examples [9]. Dynamic time warping is one technique that can be used to handle variations in the temporal behaviour of a gesture, i.e. the variation in the time it takes to perform the same gesture.

The basic tool for recognizing sequences of variable length is the Hidden Markov Model (HMM). HMMs have been successfully applied to both speech recognition and visual gesture recognition. Sequential data, as found in accelerometer-based gesture data, arise in other real world applications such as speech recognition and on-line handwriting recognition. Sequential pattern recognition has, therefore, become a very important topic in pattern recognition. HMMs have been a popular statistical tool for modelling and recognizing sequential data, particularly speech data. HMMs are probabilistic models used to represent non-deterministic processes in a partially observable domain, and are defined over a set of states, transitions, and observations [3]. The gesture recognition task corresponds to what is known as the evaluation problem, i.e. given the observation sequences and a model, what is the probability that the observed sequence was generated by the model? This is usually solved using the forward-backward algorithm, but the Viterbi algorithm can be used instead. During the training phase known examples of the gestures are used to estimate the HMM parameters. This is known as the estimation problem and is solved by the Baum-Welch reestimation procedure, which is computationally intensive. The Baum-Welch form of the expectation-maximization (EM) algorithm is used to update the parameters such that the probability that the HMM would produce the training set is maximized [9].

A Bayesian classifier is another technique that can be used for gesture recognition. Bayesian classifiers assign the most likely class to a given example described by its feature vector. A naïve Bayesian classifier is a greatly simplified version of the classical Bayes classifier, assuming that features are independent given class, that is $P(X|C) = \prod_{i=1}^{n} P(X_i|C)$, where $X = (X_1, \ldots, X_n)$ is a feature vector and C is a class. The advantages of the naïve Bayes classifier are computational efficiency, simple implementation and reliability. Unfortunately, extracting the features from the raw sensor data is computationally expensive and increases the amount of preprocessing time required.

The Kalman filer is an efficient recursive filter which estimates the state of a dynamic system from a series of incomplete and noisy measurements. They are commonly used in control theory and control systems engineering e.g. to provide information about the position and velocity of an object given only a sequence of observations of its position, each of which includes some error. Kalman filters can be compared to a HMM where the hidden states are continuous, as opposed to a HMM's discrete hidden states.

1.2 Research problem and objectives

Gesture recognition algorithms have traditionally only been implemented in cases where ample system resources are available, i.e. on desktop computers with fast processors and large amounts of memory. In the cases where a gesture recognition has been implemented on a resource-constrained device, only the simplest algorithms were considered and implemented to recognize only a small set of gestures; for example in [7], only three different gestures were recognised.

A resource-constrained device can be defined as a device with a microprocessor having a low memory footprint, small program memory and low operating frequency. Resource-constrained devices usually do not have a floating point unit (FPU) and can therefore only perform integer mathematical operations. If a hardware multiplier is not available, multiplication routines have to be implemented in software. The device used in this study was the Nokia N95 cell phone. While usually not regarded as a resource-constrained device, it does have the same limitations as other resource-constrained devices, i.e. limited amount of memory, limited processing power and a small stack and heap size. The objective was to optimize the gesture recognition algorithm such that it only requires a small amount of the device's resources. The algorithm may then be used as a user interface to a larger piece of software that will require the majority of the device's resources.

For a mobile device, the amount of power consumed during operation is of utmost importance. Vision-based algorithms require a camera-based sensor which is deemed power-hungry and resource-intensive. Inertial sensors, such as accelerometers and gyroscopes, provide a low-cost, low-power alternative for wearable application where energy resources are at a minimum. In the case of camera-based computer vision algorithms, the necessary image processing can be slow, which creates unacceptable latency for fast-moving video games and other applications [6]. Vision-based methods can also be expensive, and the sensor methods employed are restricted when used outside; whether by light levels, update rates, or the need for an external source [4]. Vision-based gesture recognition does not work when the line of sight is obstructed, and it is unsuitable for mobile or wearable applications [17].

A user must know the set of gestures that a system recognizes and gestures requiring high precision over a long period of time can cause fatigue. Therefore the gestures must be designed to be simple, natural and consistent. When inertial sensing is used, the lack of an absolute reference frame makes it impossible to track orientation relative to a fixed frame for longer than approximately five seconds [2]. Fortunately, the acceleration vectors can be used directly as input to the gesture recognition algorithm and tracking the exact orientation is not necessary. This simplifies the calibration procedure to setting the initial values of the sensor to zero, when the sensor is in a static state.

Gestures have two aspects of signal characteristics that make them difficult to recognize: segmentation ambiguity and spatio-temporal variability [10]. Segmentation ambiguity refers to not knowing the gesture boundaries, and therefore reference patterns have to be matched with all possible segments of input signals. Spatio-temporal variability refers to the fact that each repetition of the same gesture varies dynamically in shape and duration, even for the same gesturer.

The research objective of this study was to optimize a selected gesture recognition algorithm for a resource-constrained device. The various gesture recognition algorithms currently available were studied and evaluated, after which the most suitable algorithm was optimized and implemented on a resource-constrained device.

For this study, the following gesture recognition techniques (as described in chapter 4) were studied and evaluated:

- Hidden Markov Models (HMMs),

- Artificial Neural networks (ANNs), and
- Dynamic Time Warping (DTW).

1.3 Research design and methodology

The preferred research design for studies in engineering is theory-, model- or method-building, -testing and -application empirical research [11]. Related research designs that can also be considered are statistical modelling and computer simulation studies. Although these could be complete studies in their own right, it is preferred that they be used as data-gathering and data-analysis techniques for theory-, model- or method-building, -testing and -application studies.

For this study, a computer simulation study approach was initially used to evaluate and study the different gesture recognition algorithms. In the field of gesture recognition, the number of publicly available datasets based on accelerometer-based gesture data is very limited. The AcceleGlove dataset distributed with the Georgia Tech Gesture Recognition Toolkit [3] is one example of such a dataset, but consists of data obtained from multiple accelerometers, deemed too complex for this study. The dataset used in this study was gathered by making use of the Nokia N95 cell phone's built-in accelerometer. For further information on the gathering of sensor data for the dataset, see section 4.1. The dataset was used to study the performance and memory requirements of the different gesture recognition algorithms. While a gesture recognition algorithm is in itself a statistical modelling approach, the outputs of the computer simulation was used to obtain empirical data during performance analysis.

For a computer simulation study it has to be ensured that the quality of the data used and the complexity of the different gestures will still allow for a good specification of the model. Assumptions made during the design of the model must be plausible and correctly specified. For this study, the software packages that were used as a starting point for

Table 1.1: Software packages used

Algorithm	Software package
Hidden Markov Model	Georgia Tech Gesture Recognition Toolkit v. 0.1a [51]
Artificial Neural Network	Fast Artificial Neural Network v. 2.0.0 [50]
Dynamic Time Warping	Custom source code developed at Oxford University [46]

developing the gesture recognition algorithms is shown in table 1.1.

Timers and profilers are tools used to measure performance and determine if optimizations help or hinder performance [12]. A timer is a function, subroutine or program that can be used to return the amount of time spent in a section of code. It is important to make multiple measurements to ensure that results are consistent. A profiler is a tool that automatically inserts timer calls into applications. By using an profiler on an application, information is generated that summarizes timings about subroutines, functions, or even loops that were used. For further information on timers and profilers, see section 5.4.

Timers and profilers can be used to evaluate and analyze the different gesture recognition algorithms and gain deeper understanding into the performance and memory issues associated with these algorithms. Possible problem areas in the design of the different algorithms can be identified, and it can be determined which algorithms can possibly be implemented on a resource-constrained device.

The results of the computer simulation study was used to construct a model of a gesture recognition algorithm that is optimized for a resource-constrained device. This was done by selecting the algorithm that performed optimally during the computer simulation study, and optimizing each section of the algorithm with regards to processor and memory requirements. For this study a new model will not be constructed, but the existing theory and models will be refined.

To optimize the algorithm, both processor and memory requirements were studied. This was necessary as code with a larger memory footprint can sometimes be executed faster than code with a small memory footprint. Code executed sequentially is usually faster than loop-based code. As an example, consider the case where we want to multiply a variable with a constant. We can partially evaluate the multiplication routine, and generate a straight line code that multiplies by just that constant. Partial evaluation creates a tradeoff between the speed of execution, and the amount of memory required for the program [13].

The generated model was then used to implement an optimized gesture recognition algorithm on a typical real world resource-constrained device to test the feasibility of the study. This was done to ensure that no over-abstract formulations were made during the construction of the model and to empirically validate the model [14]. For details of the user application implemented on the device, see section 6.5.

1.4 Contribution

Current gesture recognition technology can be improved by making algorithms faster, more robust, and more accurate [6]. The most dramatic results in optimization are obtained by completely changing an algorithm to decrease the number of computations [12]. Algorithms can also be optimized through techniques such as partial evaluation and by profiling or timing the different sections of the algorithm to identify problem areas.

The main contribution of this study is to analyze and evaluate the different gesture recognition algorithms, to choose the one most appropriate to implement on a resource-constrained device, and then to optimize the algorithm with regards to performance and memory requirements. This was carried out through computer simulation studies and by refining a model of a gesture recognition algorithm that can be optimized for a resource-constrained device. The algorithm was then implemented on a resource-constrained device to empirically verify the model.

1.5 Challenges

The main difficulty when translating raw sensor data into recognized gestures, is the presence of noise. [18] differentiates between three types of noise:

- Sensor noise - distortion of what the actual source looks like in the signal that the sensor produces
- Sensor distance noise - distance from the actual sensor to the body part making the gesture
- Time-domain noise - sampling of sensor data may fluctuate in time, resulting in noise in the time dimension

The algorithms that process the acceleration data become slower and less effective as the number of accelerometers increases. This problem is generally known as the "curse of dimensionality" and is a common obstacle for multi-sensor systems [18]. In this study only a single-sensor system is considered.

The difficulty of implementing the algorithm on a resource-constrained device is another challenge. First the source code has to be ported to the device's operating system, while keeping in mind the limited amount of resources on the device, as well as the small stack and heap size.

Accurately comparing the different gesture recognition algorithms presents another difficulty. It has to be ensured that the testing environment is the same for the different algorithms being profiled. All the algorithms have to be written in the same programming language, to ensure that performance differences cannot be attributed to compiler or language differences. For this study, all the algorithms were implemented in the C programming language.

Chapter 2

Pattern recognition

2.1 Background on pattern recognition

According to [15], when designing a recognition system careful attention has to be paid to pattern representation, feature extraction and selection, classifier design and learning, selection of training and test samples, and performance evaluation. Pattern recognition can be one of two tasks [60]:

- supervised classification, in which the input pattern is identified as a member of a predefined class, and
- unsupervised classification, in which the pattern is assigned to a hitherto unknown class.

For this study only supervised classification is considered, as gesture recognition allows for the identification of predefined classes. The design of a pattern recognition system involves the following aspects:

- data acquisition and preprocessing
- data representation
- decision making

A well-defined and sufficiently constrained recognition problem (small intraclass variations and large interclass variations) will lead to a compact pattern recognition representation and a simple decision making strategy. It is therefore of importance that the gestures to be recognized are sufficiently different from one another in order to create large interclass variations.

According to [15], the four best known approaches to pattern recognition are:

- template matching
- statistical classification
- syntactic or structural matching
- neural networks

With template matching, the pattern to be recognized is matched against a stored template, while taking into account any temporal or spatial differences. With statistical classification, each pattern is represented as features in an n-dimensional space. The decision boundaries are determined by the probability distributions of the patterns. The goal is to have pattern vectors occupy compact and disjoint regions in the n-dimensional feature space.

With the syntactic approach, patterns are viewed as being composed of simple sub patterns, which are themselves built from yet simpler sub patterns. Neural networks consist of a network of weighted directed graphs in which the nodes are artificial neurons and directed edges (with weights) are connections between neuron inputs and neuron outputs. The most commonly used family of neural networks is the feed-forward network, which includes the multilayer perceptron and Radial-Basis Function (RBF) networks.

Another popular network is the Self-Organizing Map (SOM), or Kohonen-Network, which is used mainly for unsupervised learning. Most of the well-known neural network models are implicitly equivalent or similar to classical statistical pattern recognition methods. For example, a multilayer perceptron neural network can be viewed as nonlinear discriminant analysis.

2.2 Preprocessing

The role of preprocessing is to segment the pattern of interest from the background, remove noise, normalize the pattern, and any other operation which will contribute in defining a compact representation of the pattern [15]. By using a limited feature set that still contains all the variations in the data, the pattern representation and its classifier are simplified. This results in a faster classifier, which uses less memory. Unfortunately, performing preprocessing to determine a compact feature set is computationally expensive.

In this study the amount of preprocessing done is limited to where absolutely necessary, in order to minimize the amount of computations required. This trade-off between the computational complexity and the quality of the results should always be evaluated. The quality of the results is deemed sufficient for this specific type of problem, where the size of the feature set is compact enough not to require a large amount of preprocessing.

Feature extraction methods determine an appropriate subspace of dimensionality m in the original feature space of dimensionality d, such that $m \leq d$. One of the more popular linear feature extractors is principal component analysis (PCA) or Karhunen-Loève expansion. PCA computes the m largest eigenvectors of the $d \times d$ covariance matrix of the n d-dimensional patterns. Since PCA uses the most expressive features (eigenvectors with the largest eigenvalue), it effectively approximates the data by a linear subspace using the mean squared error criterion [15].

A feedforward neural network offers an integrated procedure for feature extraction and classification; the output of each hidden layer may be interpreted as a set of new, often nonlinear, features presented to the output layer for classification [16]. SOMs or Kohonen maps, can also be used for nonlinear feature extraction. When a multilayer feed-forward network is used for pattern classification, then the node-pruning method simultaneously determines both the optimal feature subset and the optimal network classifier.

The problem of feature selection is as follows: given a set of d features, select a subset of size m that leads to the smallest classification error. It has been argued that since feature selection is typically done in an off-line manner, the execution time of a particular algorithm is not as critical as the optimality of the feature subset it generates [15]. Most feature selection methods use the classification error of a feature subset to evaluate its effectiveness. This could be done, for example, by a classifier using the leave-one-out method of error estimation. Common feature selection methods include Sequential Forward Selection (SFS) and Sequential Backward Selection (SBS).

2.2.1 Peak-based feature extraction

Basic statistics, such as the minimum, maximum, average or covariance over a certain interval make ideal descriptors for acceleration data [18]. Peaks in the signals of the accelerometer signals can be expected to reveal even more than the basic statistics. The peaks can be detected and characterized as they occur, by making use of an activity region (defined by thresholding the size and length of a running variance).

Peak-based feature extraction [18] used a two-step algorithm that first detects an area of activity in the data, and analyzes the peaks per sensor. Unfortunately the time frame for the area of activity is relatively short, and it is not possible to track peaks over multiple dimensions. It is deemed more useful for activity recognition, e.g. recognizing whether a person is sitting, standing or walking.

2.2.2 Feature extraction method for the ReachMedia system

In this subsection the feature extraction method of the ReachMedia system [7], as described in section 2.4, is presented. For each gesture with n samples, where each sample is a vector of length 3, and each element in it represents an axis, 25 features were extracted:

- Length (1) - the number of raw signal vector samples included in the gesture, i.e. n
- Power (2-4) - the energy in each axis is calculated according to Parseval's theorem, i.e. $P(\overline{a}) = \frac{1}{n}\sum_{i=0}^{n-1} a_i^2$
- Cross correlation (5-7) - the pairwise similarity of the signal on two different axes, as measured using the correlation coefficients that are calculated for a given gesture as:

$$r(A,B) = \frac{Cov(A,B)}{\sqrt{Var(A)}\sqrt{Var(B)}} \tag{2.1}$$

  r was calculated for all pairwise combinations of the axes, i.e. xy, xz and yz.

- Inflections (8-25) - the rest of the features are the result of a signal-processing algorithm designed to find the three most significant inflections in the signal. This is done by selecting the maximal peaks in the signal.

It is interesting to note that the feature extraction method described for the ReachMedia system also makes use of peak-based feature extraction as described in the previous subsection.

2.2.3 Principal component analysis (PCA) and wavelet transforms

Mäntyjärvi et al. [23] used principal component analysis (PCA) with wavelet transforms to generate features, in order to recognize human motion with multiple acceleration sensors. Recognition was done using a multilayer perceptron classifier. PCA is commonly used to find features, or interesting directions, in terms of statistical criteria. It can be seen as an attempt to computationally find such new directions and scales for the sensors that the signals would be more discriminative.

PCA is a classical linear feature extraction method. It is based on the second order statistics of the data, in particular the eigenvalue analysis of the covariance matrix [30].

Let $\mathbf{x} = \left[x_1, x_2 \ldots x_n\right]^T$ be an n-dimensional random vector having zero mean. The task is to find an orthonormal matrix $\mathbf{V}$ of size $n \times k$, $k \leq n$ so that the reduced k-dimensional projection $\mathbf{x}' = \mathbf{V}\mathbf{x}$ retains as much of the variance of $\mathbf{x}$ as possible. The matrix $\mathbf{V}$ defines the principal directions of the projection. In practice, the principal directions and components can be calculated using the eigendecomposition $\mathbf{C} = \mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^{\mathbf{T}}$ of the sample covariance matrix $C = E(\mathbf{x}\mathbf{x}^{\mathbf{T}})$. The eigenvalues $\boldsymbol{\Lambda} = diag(\lambda_1, \lambda_2, \ldots, \lambda_n)$ determine the variance that each principal component captures. The n principal components $\mathbf{x}' = \left[x'_1, x'_2 \ldots x'_n\right]^T$ are computed by projecting the original data to the principal directions $\mathbf{x}' = \mathbf{E}^{\mathbf{T}}\mathbf{x}$ [23].

Mäntyjärvi's study used PCA for whitening or decorrelation of the signals. Whitening is a procedure where the principal components are scaled to have unit variance $\mathbf{x}^* = \boldsymbol{\Lambda}^{-\frac{1}{2}}\mathbf{E}^{\mathbf{T}}\mathbf{x}$. The data is decorrelated since $E(\mathbf{x}^*\mathbf{x}^{*\mathbf{T}}) = \mathbf{I}$.

The wavelet transform (WT) divides the original signal into wavelet coefficients c corresponding to different frequency content, thus enabling the signal to be analyzed with a resolution matched to the scale of the coefficient. In Mäntyjärvi's study, a wavelet transform utilizing a Daubechies mother wavelet order of 8 was used, which was selected for computational reasons [23].

2.3 Classifiers

The generalization ability of a classifier refers to its performance in classifying test patterns which were not used during the training stage. It is generally accepted that using at least ten times as many training samples per class as the number of features ($n/d > 10$) is a good practice to follow in classifier design [15].

The support vector classifier, or support vector machine (SVM), developed by Vapnik [52] and his colleagues at AT&T, uses the width of the margin between classes as an optimization criterion. The empty area around the decision boundary defined by the distance to the nearest training patterns, called the support vector, define the classification function.

A classifier is trained during a number of epochs. One epoch means going through the entire training data once. The classification error or simply the error rate, P, is the ultimate measure of the performance of a classifier. Performance, in this instance, is the ability of the classifier to correctly classify a pattern into a class, and not its computational performance. The error rate is estimated by dividing all the available data samples into a training set and a test set. The training and test sets should be sufficiently large in order to predict future classifier performance, and must be independent. Various methods are used here, including the leave-one-out method and the cross-validation method.

2.4 Current implementations

Choi et al. [17] used accelerometer data acquired from a mobile phone's built-in accelerometer. They were able to recognize digits from 1 to 9 and five symbols written in the air. During their experimental study, they were able to achieve a 97.01% average recognition rate for a set of eleven gestures. The recognition rate was cross-validated from a data set of 3082 gestures from 100 users. This was done using a BN-based approach, with gesture recognition done on a PC connected to the mobile phone.

ReachMedia [7] is a system for providing information about everyday objects. It makes use of an RFID reader to detect objects that the user is interacting with. Continuous gesture data is then used as input to interact with the system. Gestures were used for the project because they can be made quite minimal and therefore socially acceptable, while retaining ease of use. A discretized Naïve Bayes classifier was used that showed equivalent results to a HMM approach. A simple axis-by-axis variance window with preset thresholds, introduced by [2], was used to detect when a gesture begins and ends. A 16-sample window with a start threshold and a stop threshold was used. When the variance of one ore more axis passed the start variance threshold, the gesture was recorded. Recording was stopped once the magnitude of all the axes went below the stop variance threshold.

Huang et al. [21] have developed a Mandarin-language speech recognizer that can distinguish between about 400 000 words, running on a Compaq iPaq 3600 PDA with a SA-1110 206MHz StrongARM (2.1 MIPS) processor, with a HMM-based classifier. It takes up about 200KB of memory, and the models and tables take about 2MB. While it is performing speech recognition it uses an additional 1MB RAM. Nearly all the speech recognition engines on the market today are based on HMMs, which are used to represent how phonemes and allophones are pronounced and how fast they are spoken. Neural network-based engines tend to be better than HMMs for picking out discrete words, but extensive training is required up front. For recognizing continuous speech, HMMs are deemed to perform better than neural networks [21].

The Nintendo Wii games console has received a lot of press recently, and is currently one of the most popular games consoles in the world. It makes use of an accelerometer-based remote control to generate gesture data that can be used as input to a video game. AiLive Inc., who develops the gesture recognition technology for the Wii, state that their real challenge was to get it working in real-time using around 5% of a modest CPU and using no more than a few hundred kilobytes of memory [53]. The feature space was defined with duration (how long a motion lasts) on the X-axis and maximum amplitude (the maximum magnitude of the motion sensor data) on a two-dimensional graph. Values are plotted for measurements in a three-dimensional space. This is similar to other feature extraction methods, which use peaks in the accelerometer data as features. They use a technique they call *Context Learning* to create and evaluate a large collection of nonlinear separators. Other features for a specific class include the maximum and minimum time for a gesture, the initial orientation of the controller and the initial impulse (e.g. going up, level or going down).

Researchers at Georgia Tech University [22] have developed a system using a vision-based sensor and multiple accelerometers for mobile sign language recognition. The accelerometer feature vector consist of x, y and z values for accelerometers on the left wrist, right wrist and torso. Gestures in the vocabulary were represented by two different HMM topologies. Short words (e.g. my, me, talk, exit, calibrate) were represented with a 5-state left-to-right HMM with self-transitions and 1 skip-state. Longer words (e.g. computer, helps) were represented with a 10-state left-to-right HMM with self-transition and 2 skip-states. They made use of the leave-one-out testing mode and collected statistics for the training and testing runs. Gesture recognition was performed on a PC and not on a resource-constrained device.

Pylvänäinen has employed an accelerometer-based gesture recognition algorithm using continuous HMMs, with movements recorded using a 3D accelerometer embedded in a handheld device [25]. The data was recorded on a mobile device running the Symbian platform, but gesture recognition was performed on a desktop PC. The data polling was timed using Symbian methods and was subject to variability due to the multitasking nature of the operating system. The actual sampling rate was approximate and variable, but in the neighbourhood of 30Hz. A left-to-right HMM with continuous normal output distributions was used. The output distributions were assumed to have diagonal covariance matrices. The performance of the recognizer was tested on a set of 10 gestures, 20 gesture samples from 7 different persons, resulting in a total of 1400 gesture samples. Every model for each of the 10 gestures had 8 states. The data was normalized by rotating the data so the estimate of the gravitational constant, g, is pointing to the negative y-axis. 99.76% accuracy was obtained with user-independent testing. Pylvänäinen argued that an extensive set of gestures (i.e. more than 10) becomes impractical due to users having to learn all the different gestures.

Mäntylä et al. made use of a self-organizing Kohonen mapping scheme for recognizing static gestures, and a HMM used to recognize dynamic gestures. During preprocessing each vector component was filtered and normalized separately. Low-pass filtering was carried out with a fourth-order IIR Butterworth filter. A 3-dB cut-off frequency of 2.5 Hz was used for dynamic gestures. The variance of each component was normalized to one, and the mean to zero [26]. They made use of of the Baum-Welch and Viterbi algorithms for the training and recognition tasks respectively. Acceleration data was collected at a sampling rate of 100 Hz and recognition was performed on a laptop PC. They noted the following factors causing errors when more than one user is involved:

- Dynamical differences (intensive vs. phlegmatic)
- Temporal differences (slow vs. fast)
- Physical dimensions of the testee (length of body and reach of the hands)
- Standing pose of the testee

They did note, however, that temporal variations were handled well by the HMM models.

Westeyn et al. developed a system to recognize mimicked autistic self-stimulatory behaviours using HMMs. Data was sampled using three wireless sensors, each with two perpendicularly mounted dual-axis accelerometers, at a rate of 100 Hz. An hour of data collection produced approximately 36 MB of ASCII text data. They made use of left-to-right topologies for the HMM models, with self transitions and a single loop back to a previous state, as well as one allowable skip at each state. At each state, a nine-element vector, representing the 3-axis readings from each of the three sensors can be observed [28]. They defined the accuracy of the system as follows:

$$Accuracy = \frac{N - S - D - I}{N} \tag{2.2}$$

where substitution errors (S) occur when the system incorrectly classifies a behaviour, insertion errors (I) occur when the system recognizes an instance of a behaviour that did not occur, deletion errors (D) arise when the system fails to recognize the occurrence of a behaviour within the stream of data, and N represents the total number of examples. It should be noted that insertion and deletion errors only occur during continuous recognition [28].

Bailador et al. made use of continuous time recurrent neural networks (CTRNN) for real-time gesture recognition [29]. CTRNNs exhibit rich dynamic behaviour that is useful in gesture recognition, but also have a relatively low computational cost that is required for real time systems. Accelerometer data was captured on a desktop PC connected to a tri-axial sensor module via Bluetooth. For these types of neural networks, a global optimization of the network parameters using genetic algorithms can be performed. The gestures were isolated by resting the hand in the same position between gestures. The gestures used to analyze the performance of the method was also used in this study and can be seen in figure 4.2 in section 4.1.

Murakami et al. used a discrete-time recurrent neural network for gesture recognition [35]. Absolute as well as relative hand position were relied on for gesture recognition, requiring an expensive data glove for gesture input. An Elman recurrent neural network was used to process the time-series data. The network architecture makes use of a hidden layer that feeds back on itself through a context layer. The neural network was trained using the backpropagation algorithm. The input layer consisted of 16 nodes - 10 for bending data, 3 for coordinate angles and 3 for positional data. The hidden layer consisted of 150 nodes and the output layer of 10 nodes. Each of the output nodes corresponded to a word - the node in the output layer with the greatest value was recognized as that specific word.

D. Xu made use of a data glove and a neural network to recognize hand gestures, as input to a virtual reality based driving training system for a Self-Propelled Gun (SPG) [36]. A feed-forward neural network can represent an arbitrary functional mapping, so it is possible to map raw data directly to the required hand gestures. A feed-forward neural network with a single hidden layer was used to recognize various hand gestures. A back-propagation method with a variable learning rate was used for training, due to inherent characteristics like:

- simplicity,
- robustness, and
- minimal memory requirements.

These factors are also very important when selecting a gesture recognition algorithm for a resource-constrained device. An adaptive learning rate was selected to compensate for the slow training times associated with back propagation networks. The learning rate was initially set to be very small, and then dynamically increased every training epoch. If the error rate increased, the last epoch was discarded and the learning rate decreased.

The number of nodes in the hidden layer was selected empirically. Similar to determining the learning rate, the number of nodes in the hidden layer was initially set to be small, and then increased after every epoch, until the network was trained to an acceptable error level. Unfortunately, the system only recognizes static hand gestures, and dynamic gestures were left as future work.

Ko et al. [45] implemented a system to perform context recognition in multisensor systems using DTW [45]. The systems uses DTW to recognize multimodal sequences of different lengths, embedded in continuous data streams. Context recognition is the extracting, fusing and converting of relevant data from multiple sensors for situation awareness. The system was able to recognise multimodal sequences of different lengths generated from multiple sensors with both discrete and continuous outputs. They found that DTW complexity is linear with respect to the number of dimensions in the data and hence can deal with high dimensional time varying data.

The implementation that inspired this study was the ReachMedia system developed by [7], indicating that performing gesture recognition on-board a resource-constrained device is feasible. The implementations by [22], [25], [26] and [28] all indicated that an HMM is well suited to accelerometer-based gesture recognition and was consequently included in the evaluations done in this study. The neural network implementations by [29], [35] and [36] also indicated that ANNs should form part of the evaluations done in this study. DTW has not been widely used for gesture recognition in the past, but the context recognition implementation by [45] showed promising results. It was decided to include DTW as part of the evaluation to see if it will be feasible to use in a gesture recognition algorithm.

Chapter 3

Designing a gesture recognition system

Designing a real-time gesture recognition system is a complex task that involves many issues such as algorithm design, processing speed, and system architecture [20]. During the design phase, simulation tools are used to measure the performance of different algorithm elements and optimize the algorithm to improve processing speed. In the following sections, the various pattern recognition techniques are considered for designing a gesture recognition system.

3.1 Bayesian networks

Bayesian networks are directed acyclic graphs (DAGs) where the nodes are random variables, and certain independence assumptions hold [32]. The arcs in a Bayesian network specify the independence assumptions that must hold between the random variables. These independence assumptions determine what probability information is required to specify the probability distribution among the random variables in the network. A causal graph of a Bayesian network is shown in figure 3.1. The nodes denote the different states, while the arcs are the causal connections. A recognition algorithm based on Bayesian networks (BNs) has been applied to recognize trajectories on a 2-dimensional plane [54], by making use of both acceleration and angular velocity signals. Choi et al. sug-

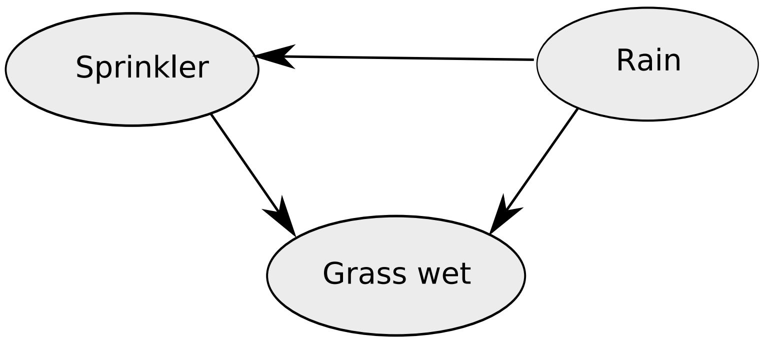

Figure 3.1: Causal graph of a Bayesian network

gested a simple attitude estimation technique to estimate the trajectories of hand signals by making use of acceleration signals only [17]. It was also found that it is possible to recognize gestures from a tri-axis accelerometer without making use of trajectory estimation.

Gesture recognition using BN models was implemented by [17] as follows:

- The raw signal is normalized by removing gravity acceleration components and by rescaling the signal's amplitudes.
- Feature points in the signal are detected, where feature points are defined as local minima or local maxima points where the signal values are minimal or maximal within a predefined time interval. This is done to minimize temporal differences among different users.
- The normalized feature sequence is matched to gesture models.
- BN-based recognition models are trained with the collected data. The model parameters are estimated from the data in the criteria of maximum likelihood estimation (MLE).

The gesture model is composed of basic primitive models with their dependencies, where the primitive models represent the portion of sample points between feature points which have a similar trend. Examples of such trends are monotonic increasing and decreasing [17]. Their dependencies are represented by a conditional Gaussian model.

3.2 Neural networks

Artificial Neural Networks (ANNs) are mathematical functions and, as such, naturally take continuous or discrete numeric data as inputs. The standard forms of ANNs should be presented with the smallest set of inputs that contain relevant information. Although ANNs have some capability to distinguish relevant from irrelevant inputs, this can only be done reliably with large volumes of data consisting of thousands of samples [33]. Most standard forms of ANN cannot cope with missing inputs, and an ANN is usually applied in such a way that all the required information is always available. ANNs generally fall into one of two categories:

- those that create a model by memorizing some part of the data they learn from,
- and those that, like the Multilayer Perceptron (MLP), create a model by building an abstract representation of it.

The former network typically have a high computational efficiency while learning, while the latter networks tend to have extremely low computational efficiency while learning. The space required to store the latter ANN is very small, but the storage requirements of that work by memorizing can be large. As an example of typical storage and computational requirements, an MLP with six inputs and one output will require less than 1 kilobyte of storage, the execution of six exponential and reciprocal functions to query, and the execution of perhaps 100 million exponential and reciprocal functions to learn.

A neural network such as an MLP is essentially a complex nonlinear function with a number of adjustable parameters that can be changed to control its shape. The process of training the network is one of adjusting its parameters so that the function it represents takes on a desired shape. Input selection is the most labour-intensive part of developing a neural network application, and finding a small set of inputs that is rich in relevant information is crucial to success [39].

In Mäntyjärvi's activity recognition study (also described in section 2.2.3) three multilayer perceptron (MLP) neural networks using backpropagation learning were used for classification [23]. The MLP networks were three-layer networks consisting of 24 input neurons, 43 hidden neurons and 4 output neurons. All neurons used logistic scaling functions. The initial parameters set the learning rate to 0.1 and the momentum to 0.1. A momentum-based weight update was used in training.

Neural networks should be designed to be invariant to various transformations of the input data. The structure of the neural net can be designed such that its output is always invariant to certain transformations [24]. The number of required connections will be prohibitively large for implementation on a resource-constrained device. Alternatively, the neural net can learn all the different transformations by presenting these transformations during the learning phase. This would require a vast amount of training data during the learning phase. Another option is to present input features to the neural net which are invariant to the transformations themselves. By making use of feature extraction methods the input features can be made invariant to the transformations, for example spatial or temporal changes.

The main disadvantage of using invariant feature spaces is the need to first calculate the features before the classifier can be employed [24]. Invariant feature spaces that are computationally expensive to calculate are problematic for implementation on a resource-constrained device. Invariant feature spaces which have been used with neural networks include wedge-ring samples of the magnitude of the Fourier transform, the magnitude of the Fourier transform in log-polar coordinates and moments [24]. Moments are problematic when there is noise in the system, while Fourier transforms are not invariant to all possible transformations.

Accelerometer-based gesture recognition is similar in nature to speech recognition. Both

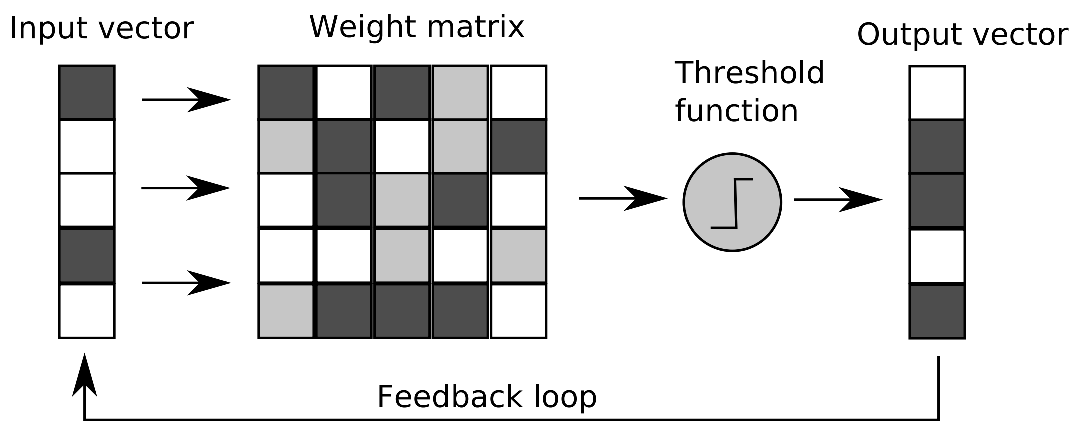

Figure 3.2: Binary Hopfield network architecture

consist of time-series dynamic signals. Traditionally speech recognition algorithms make use of hidden Markov models, Bayes decision theory, vector quantization (VQ), and auto-regressive (AR) modelling [30]. It is necessary to investigate these techniques to see if they can be applied to accelerometer-based gesture recognition as well. A specific type of neural network, called a recurrent neural network (RNN) has been used in the speech recognition field, and is suited to the modelling of the temporal structure of accelerometer signals. A continuous-time RNN gesture recognition implementation is described in section 2.4. With an HMM the temporal information of the signal is memorized using a state model, whereas with a RNN the information is modelled in a data-recurrent structure.

A Hopfield network is a type of recurrent neural network that is defined as a feedback dynamical system. A Hopfield network consists of an input vector, a weight matrix, a threshold node and an output vector. A recurrent network is considered stable when it converges to a fixed point. When a fixed point is input into a dynamical system, the same point results as output. Hopfield networks are considered stable, but usually there will be more than one fixed point. The architecture of a binary Hopfield network is shown in figure 3.2. Depending on the input vector, the Hopfield network can give as output one of the fixed points in the output vector [31].

One must always consider how much data needs to be collected. In practice, provided that the number of hidden neurons is kept small, good performance can be achieved with as few as 10^I training samples for a network with I inputs. Although the gradient descent optimization algorithm is typically used to fit an MLP to the training samples, a rarely applied technique called the perturbation search can also be used [39]. The perturbation search does not require gradient information, is easier to program, easier to understand and easier to apply (for example, it is guaranteed to be stable). It allows integer versions of networks (that are applicable to resource-constrained devices) to be optimized directly, avoiding problems that can result from the conversion of floating-point networks to integer form.

The basic perturbation search can be summarized as follows:

- Measure the performance of the MLP
- Perturb the MLP's parameters by adding a small amount of random noise to each one, and remeasure its performance
- If the performance of the MLP deteriorated, restore the parameters to their original values
- Repeat this process until some stopping criterion is met

3.3 Hidden Markov models

Noisy processes can be successfully estimated using stochastic techniques. Hidden Markov Models (HMMs), in particular, have been successfully used to track finite-state processes based on noisy evidence. For example, in speech recognition, HMMs track the production of words as movement through a space of phonemes and sounds, based on very noisy evidence but very concrete transition rules. In general, a Markov model can be visualized as a a Finite State Machine (FSM) with probabilistic edges. In a *hidden* Markov model, the state of the process is not directly observable, so one can never be sure of the current state of the process. Instead, the HMM includes a probability function that matches states with some type of observable evidence - this allows one to estimate the current state based on a history of evidence observations.

HMMs have been applied successfully to speech recognition and visual gesture recognition. These methods translate almost directly to accelerometer-based gestures [25]. In speech recognition, feature extraction is performed by transforming the speech data into the frequency domain and then decorrelating the data. Feature extraction reduces the amount of information, both in terms of time resolution and in terms of dimensionality. As the computational complexity of HMM decoding is linearly dependent on the number and dimension of feature vectors, feature extraction increases efficiency.

Gestures can be regarded as temporal feature trajectories with temporal and spatial variations which can be successfully recognized by HMMs, with the most common HMM topology being the left-to-right topology [26]. In ergodic or fully connected HMMs every state of the model can be reached (in a single step) from every other state of the model [41]. The left-right type of HMM has the desirable property that it can readily model signals whose properties change over time in a successive manner e.g. speech. The fundamental property of all left-right HMMs is that the state-transition coefficients have the property $a_{ij} = 0$, $j < i$. This implies that no transitions are allowed to states whose indices are lower than that of the current state. The initial state probabilities have the property π =(0, $i <> 1$ and 1, $i = 1$) because the state sequence must begin in state 1 (and end in state N).

A gesture is a specific, intentional action by a human in which part of the body is moved in a predefined way, indicating a stochastic event. Chambers et al. [27] considers a gesture as a stochastic process that exists at multiple levels in a hierarchy - simple, discrete movements correspond to gestures at one level, and combinations of these movements as gestures at a higher level. By utilizing multi-layer HMMs to model complex gestures as a hierarchy of sub-gestures, it was possible to recognize a sequence of sub-gestures, without the need for learning separate HMMs for each sequence [27]. Designing an HMM is considered more difficult than designing multilayer perceptron neural networks, but the superior performance in some complex tasks may justify their use.

3.4 Support vector machines

A Support Vector Machine (SVM) is a type of neural network where a nonlinear transformation is performed on the input feature vector into a high-dimensional space, as a new feature vector $\varphi(\mathbf{x}) = [\varphi_1(\mathbf{x})\varphi_2(\mathbf{x})\ldots\varphi_p(\mathbf{x})]$. If $\varphi(\mathbf{x})$ is the nonlinear transformation, and $\mathbf{x}$ is the input feature vector, the output y can be calculated as

$$y(\mathbf{x}) = \sum_{k=1}^{p} w_k\varphi_k(\mathbf{x}) + b = \varphi(\mathbf{x})w^T + b, \tag{3.1}$$

where $\mathbf{w} = [w_1w_2\ldots w_p]$ is the $1 \times p$ weight vector and b is the bias term. Mapping a low-dimensional feature into a higher-dimensional feature space is likely to make the resulting feature vector linearly separable [30]. By using φ as a feature vector we should be able to obtain better pattern recognition results. Commonly used kernel function for support vector machines include radial basis functions and two-layer perceptrons. An example of an SVM neural network structure can be seen in figure 3.3.

3.5 k-Nearest neighbours

The k-Nearest neighbours (kNN) algorithm is a statistical technique that learns by memorizing the data it is presented, thereby using the data itself as a model of what it has learned [33]. A kNN algorithm computes the Euclidean distance between the training data and the input data. As the kNN algorithm does little more than store the data that it is required to learn, it offers very high computational efficiency when learning. This

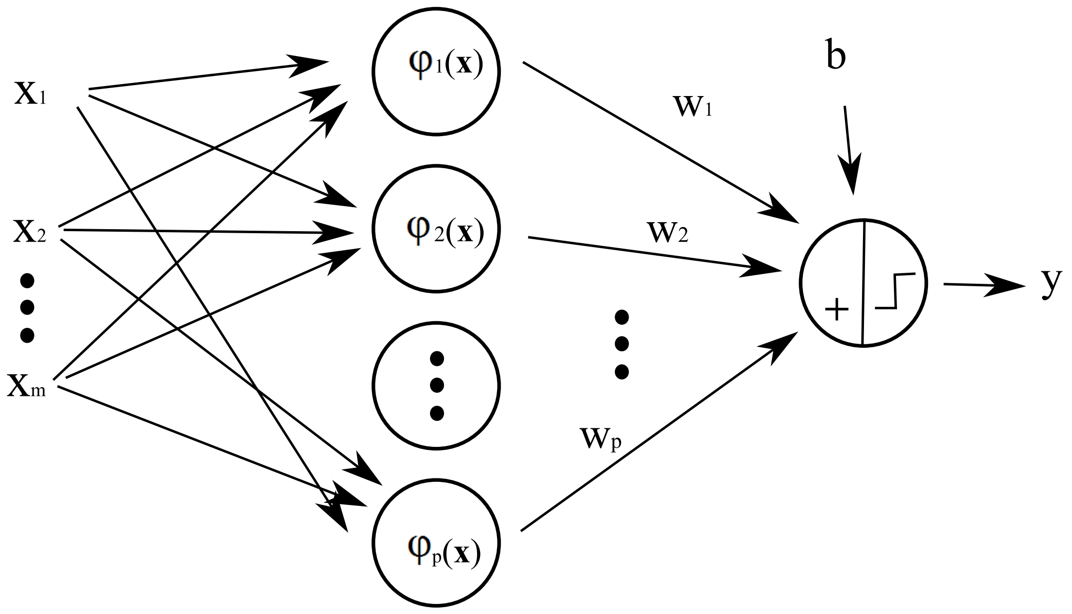

Figure 3.3: An SVM neural network structure

advantage comes at the cost of increased computation during the query phase when the new input must be compared to the data that has been learned. A realistic kNN would operate on feature vectors rather than the raw data, but this requires an additional computationally intensive preprocessing stage.

3.6 Dynamic time warping

Dynamic time warping is a general time alignment and similarity measure for two temporal sequences, that was introduced by Sakoe and Chiba [47]. A distance matrix is computed for the two sequences and a minimum cost path is determined along the diagonal of the matrix. Similar to kNN, the dynamic time warping algorithm only has to store the data that it is required to learn. The training and recognition procedures in dynamic time warping are potentially much simpler and faster than HMMs [45]. Dynamic time warping is described in more detail in section 4.4.2.

Chapter 4

Implementation

4.1 Gathering of sensor data

Sensor data was collected using a Nokia N95's embedded 3-axis STMicroelectronics LIS302DL accelerometer. The Symbian 3rd Edition SDK's Sensor API was used to gather raw sensor data using an interrupt-driven sampling method. The data was filtered using both a digital low-pass filter (LPF) and a high-pass filter (HPF). A LPF with a single pole of $K_1 = 0.97$ was applied to the x- and y-axis, with a single $K_2 = 0.9$ pole applied to the z-axis. A HPF with a single pole of $K_3 = 0.75$ was also applied to the z-axis. In figure 4.1 the raw sensor data gathered from the mobile phone's accelerometer is shown for all the three axes.

The accelerometer was calibrated by placing the mobile phone on a flat surface and resetting all three axes to zero. The required amount of training samples was determined empirically. A HMM was trained with two different gestures, starting with one training sample each and increasing the amount of training samples until the HMM could distinguish between the two gestures. For the HMM to differentiate between two gestures, at least three training samples per gesture were required. A total of 8 gestures with 10 training samples per gesture were collected. As the DTW algorithm is essentially a type of template-matching technique, only one training sample per gesture was required for

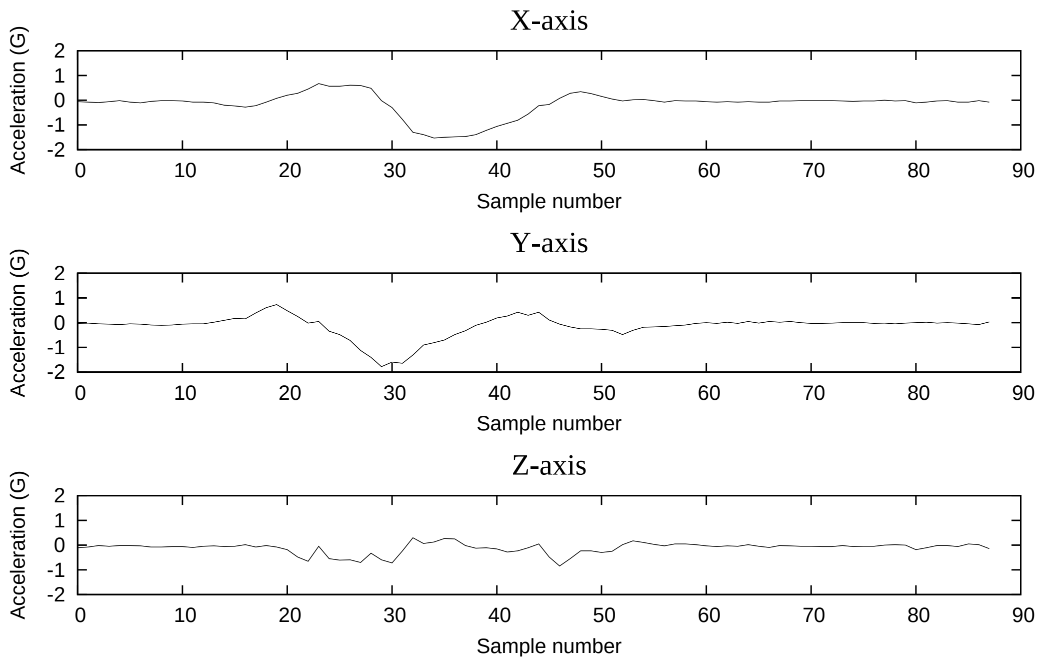

Figure 4.1: Raw sensor data of a *Left Circle* gesture sampled from the Nokia N95's accelerometer

the DTW algorithm to perform the gesture recognition correctly. The 8 gestures used in this study can be observed in figure 4.2.

It is difficult for a person to memorize a large amount of gestures. Learning gestures is analogous to learning a new language. For this reason only 8 different gestures were utilized. The main application of this technology on mobile phones, where typically only one built-in accelerometer is available, is considered to be for user interfaces and as gesture input to mobile games. Where more gestures are required, for example in sign language recognition, more sensors will be required and a different kind of approach will have to be used. Most of the research in the field of gesture recognition use a limited set of gestures, unless the gesture recognition technology is well established, as is the case with HMM-based recognition using multiple sensors.

A comparison of two different data streams of the same gesture (*Left Circle*, item 1 in figure 4.2), is shown in figure 4.3. At first glance there seems to be no similarity between

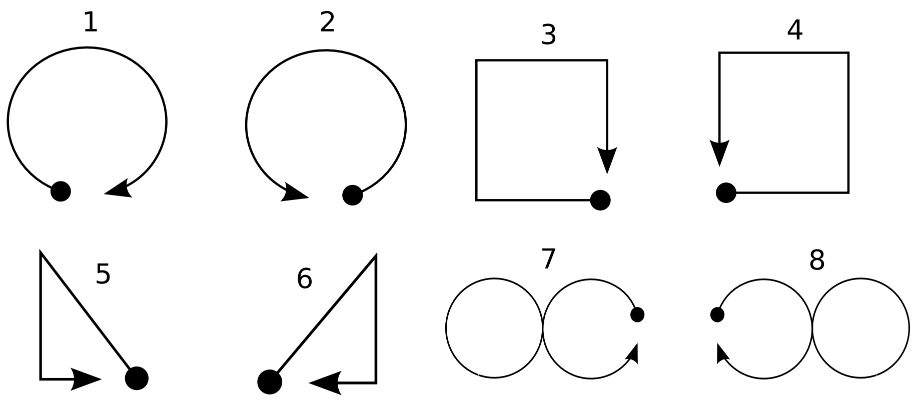

Figure 4.2: Gestures used in this study (obtained from [29])

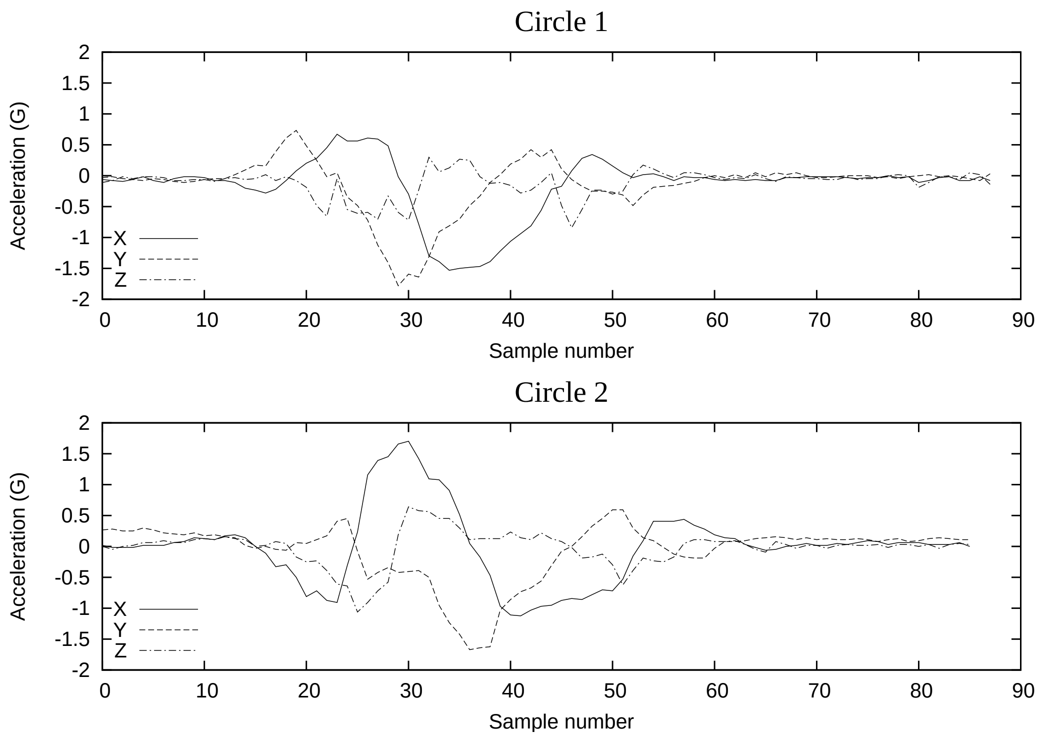

Figure 4.3: Comparing two sensor data streams of the same gesture (*Left Circle*)

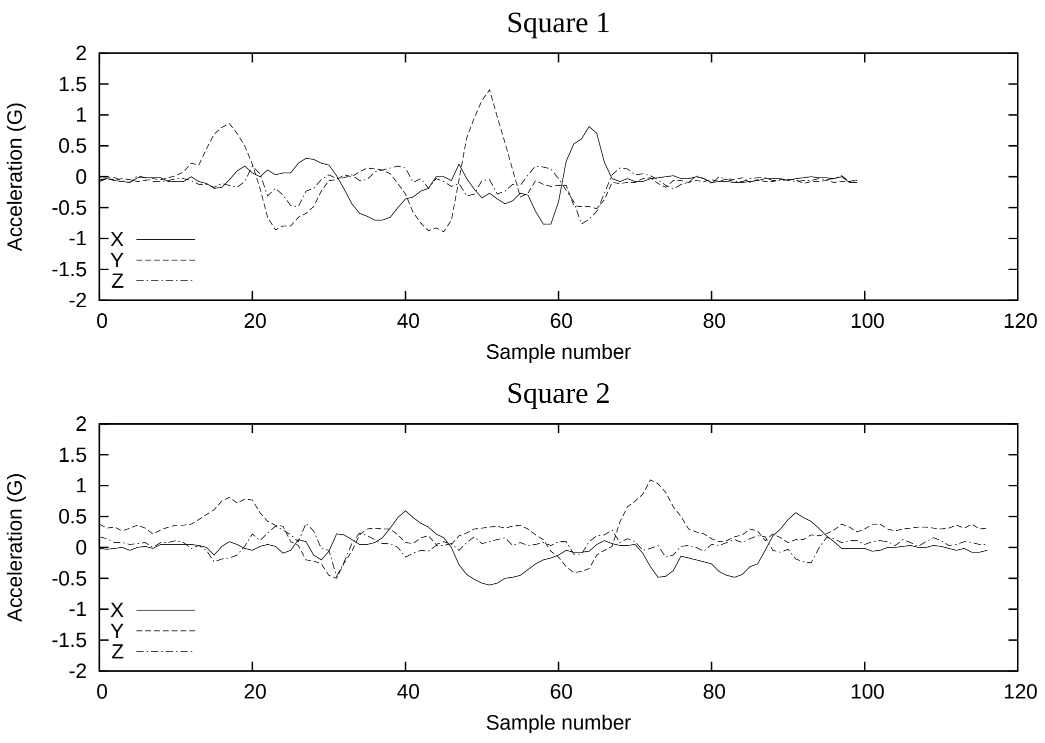

Figure 4.4: Comparing two sensor data streams of the same gesture (*Left Square*)

the two data streams. On further inspection one can see that, although they differ in length and amplitude, there is first a sharp increase and then a sharp decrease in the acceleration values of both gestures. Figure 4.4 shows a comparison between two data streams of the *Left Square* gesture (item 3 in figure 4.2). Once again, the length and amplitude of the gestures differ, but a similar pattern can be distinguished in the data.

4.2 Hidden Markov model

4.2.1 Preprocessing

For the Hidden Markov model algorithm, a gesture recognition toolkit developed by the Georgia University of Technology, called GT2K [3], was utilized. GT2K makes use of the HTK Speech Recognition Toolkit developed at the University of Cambridge [42]. Recognition was performed by the `HVite` utility in HTK that uses the Viterbi algorithm to solve the decoding problem, i.e. given the observation sequence O and the model M_i, a corresponding state sequence X must be chosen that is optimal in some sense. The input to the `HVite` executable is the following:

- The list of commands that can be recognized
- The names of the gestures corresponding to the commands
- The trained HMM
- A word lattice generated by HTK which links the gestures to the HMM
- An `-A` parameter to print the current command line parameters
- A `-T 1` parameter to enable tracing

The raw data was preprocessed by using a utility called *standalone_prepare* in GT2K, that transforms the raw sensor data into the format required by HTK.

4.2.2 The algorithm

A Hidden Markov Model with a feature vector length of 3 was used, where each feature vector consists of the x-, y- and z-axis acceleration values. The feature vector length is the number of elements in the feature vector. This is also called the number of observations per state for the HMM.

A 4-state HMM with no skip states was used at first. When the HMM performed suboptimally (see section 6.1.1), it was decided to make use of an 8-state HMM with no skip states. The minimum variance allowed during training was 0.001. The validation used for training and testing the algorithm was leave-one-out validation. This validation method was also used to test the performance of the ANN and DTW algorithms.

4.3 Neural network

4.3.1 Preprocessing

Applying a Fourier or wavelet transform to the input data before feeding it into the neural network classifier would create a compact representation of the pattern to be recognized. The increase in computational complexity is deemed unnecessary for a small amount of gestures. Should a larger gesture set be used, this type of preprocessing technique can be considered for improving pattern representation.

The length of the data vector that is used as input to a discrete wavelet transform (DWT) must be a power of two. The same constraint applies to fast Fourier transforms (FFTs). With FFTs the individual sine and cosine functions are localized in frequency, while with DWTs the individual wavelet functions are localized in both frequency and space [48]. Resampling will still be required for these transforms, in order to make the data vector length a power of two, further increasing the computational complexity of the algorithm. For this study resampling will still be performed, in order to create a fixed amount of inputs for the neural network, but without transformations on the data.

As a feedforward backpropagation neural network only operates on a fixed amount of inputs, the data had to be resampled to be used as input to the neural network. A simple resampling algorithm as described in [59] was implemented:

Let M be the original number of points in the path and N the required number of points. To resample, one first calculates the total length of the M-point path. Dividing this by $(N-1)$ results in the length of each increment I, between N new points. Then the path is subsequently traversed so that when the distance covered exceeds I, a new point is added through linear interpolation. At the end of this step, all gestures will have exactly N points.

The resampling algorithm resamples W points into n evenly spaced points. The pseudo code for the resampling algorithm is as follows:

```
RESAMPLE(points, n)

 1. $I \leftarrow \frac{PATHLENGTH(points)}{n-1}$
 2. $D \leftarrow 0$
 3. $newPoints \leftarrow points_0$
 4. **foreach** $point\ p_i\ for\ i \geq 1\ in\ points$ **do**
 5.     $d \leftarrow DISTANCE(p_{i-1}, p_i)$
 6.     **if**$(D + d) \geq I$ **then**
 7.         $q_x \leftarrow p_{i-1}(x) + \left(\frac{I-D}{d}\right)\left(p_i(x) - p_{i-1}(x)\right)$
 8.         $q_y \leftarrow p_{i-1}(y) + \left(\frac{I-D}{d}\right)\left(p_i(y) - p_{i-1}(y)\right)$
 9.         $APPEND(newPoints, q)$
10.         $INSERT(points, i, q)$ $//q\ will\ be\ the\ next\ p_i$
11.         $D < 0$
12.     **else** $D \leftarrow D + d$
13. **return** $newPoints$
```

```
PATHLENGTH(W)

 1. $d \leftarrow 0$
 2. **for** $i$ **from** $1$ $to$ $|W|$ **step** $1$ **do**
 3.     $d \leftarrow d + DISTANCE(W_{i-1}, W_i)$
 4. **return** $d$
```

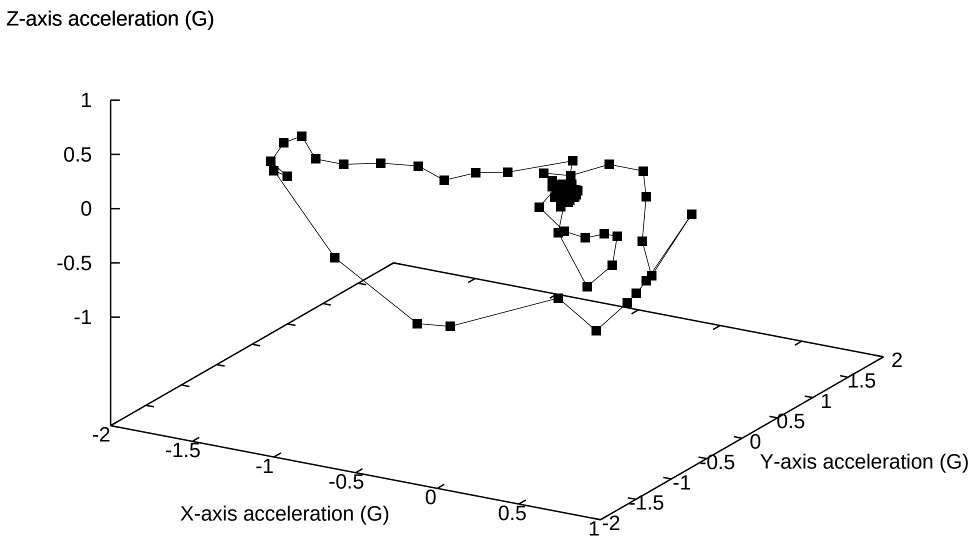

Figure 4.5: 3-dimensional plot of raw sensor data

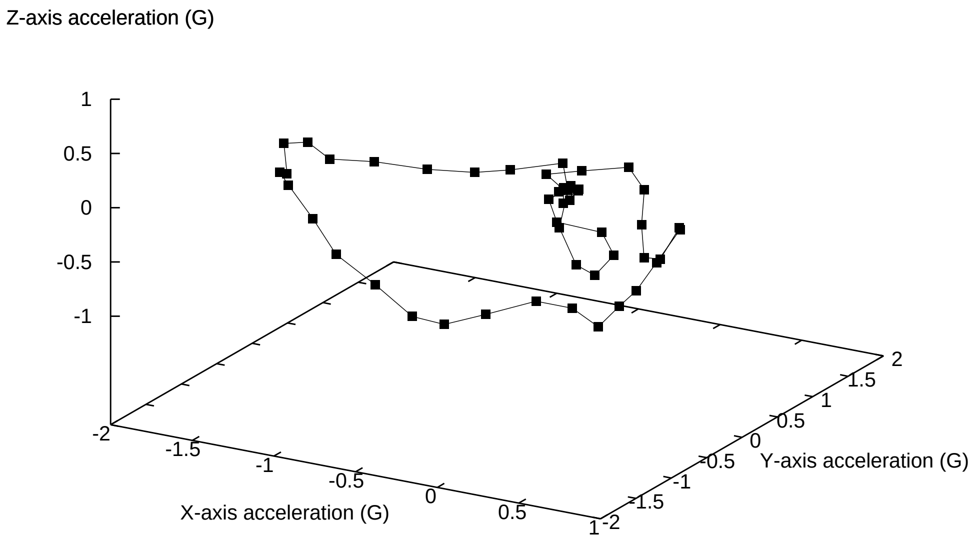

Figure 4.6: Sensor data resampled to N = 50 points

The original two-dimensional algorithm was rewritten in order to cater for the three-dimensional nature of the accelerometer input data. Comparisons of the original raw sensor data and resampled data can be seen in figures 4.5 and 4.6.

This algorithm can be used to either downsample or upsample the data, depending on the requirements. Regarding the ANN used in this study, the sampled gestures were down-sampled to 50 points per axis. With 50 points each for the three axes x, y and z, a total of 150 points were used as input to the neural network.

4.3.2 The algorithm

Many practical problems require no more than one hidden layer in a neural network [49]. Neural networks with one hidden layer can approximate arbitrarily with any functions which contain a continuous mapping from one finite space to another.

Choosing the number of hidden neurons is a very important part of designing the overall neural network architecture. Underfitting occurs when there are too few neurons in the hidden layers to adequately detect the signals in a complicated data set [49]. Using too many neurons in the hidden layers can result in overfitting, that occurs when the neural network has so much information processing capacity. In this case the limited amount of information in the training set is not enough to train all of the neurons in the hidden layers. There are a number of heuristics for determining the correct number of neurons in the hidden layer:

1. The number of hidden neurons should be between the size of the input layer and the size of the output layer.
2. The number of hidden neurons should be two thirds the size of the input layer, plus the size of the output layer.
3. The number of hidden neurons should be less than twice the input layer size.

Table 4.1: Configuration parameters of the neural network

Number of input layers	150
Number of output layers	8
Number of hidden layers	1
Number of hidden neurons	108
Desired error	< 0.001

Considering a neural network with a input layer size of 150 and a output layer size of 8, a hidden layer of 108 neurons fulfills all three the abovementioned requirements. To improve the accuracy of the neural network, forward selection methods and backward selection methods may be used. These methods are described in section 2.2.

A neural network library called Fast Artificial Neural Network (FANN) library [50] was used as a starting point for developing the ANN-based gesture recognition algorithm. A standard fully connected backpropagation neural network was created. A bias neuron was placed in each layer (except the output layer), and this bias neuron was connected to all neurons in the next layer. When running the network, the bias nodes always emit ones. The configuration parameters of the neural network can be seen in table 4.1.

The activation function used for both the hidden layer and the output layer is a symmetric sigmoid function ($tanh$). It is one of the most applied activation functions for a neural network. It is defined as

$$y = tanh(sx) = \frac{2}{1+e^{-2sx}} - 1 \qquad -1 < y < 1, \tag{4.1}$$

where x is the input to the activation function, y is the output, and s is the steepness. An example of a symmetric sigmoid activation function can be observed in figure 4.7.

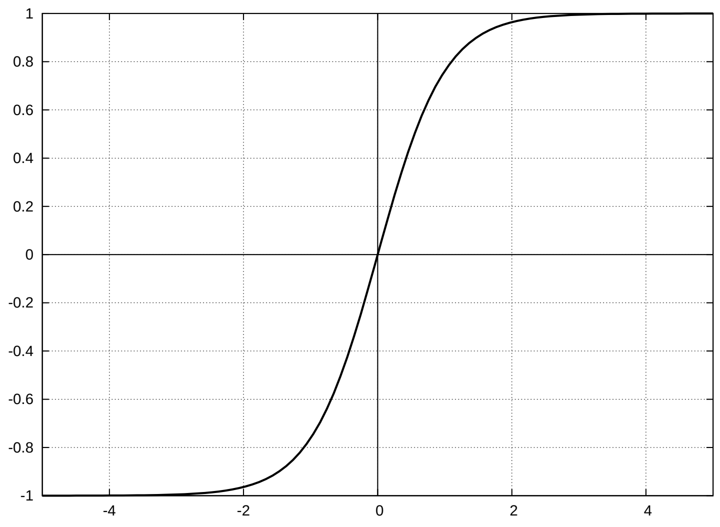

Figure 4.7: A symmetric sigmoid activation function

To train the neural network for a desired error of less than 0.001, around 25 epochs were needed. In figure 4.8 the average error per pattern is plotted, where $J = \sum_{n}^{p=1} J_p$ is the sum over the errors on n individual patterns, also known as the total training error [61].

4.4 Dynamic time warping

4.4.1 Preprocessing

For the dynamic time warping algorithm, no preprocessing is required. The algorithm is used directly to match the test gesture against the reference gestures.

4.4.2 The algorithm

The DTW algorithm used in this study was implemented in C by Andrew Slater and John Coleman [46] at Oxford University Phonetics Laboratory. The DTW non-linearly wraps one time sequence to match another given start and end point correspondence. DTW was introduced by Sakoe and Chiba [47] in a seminal paper in 1978. DTW has been extended to deal with unknown start and end points in a continuous data stream. More recent research has focused on applying it to data mining from one-dimensional time series, as

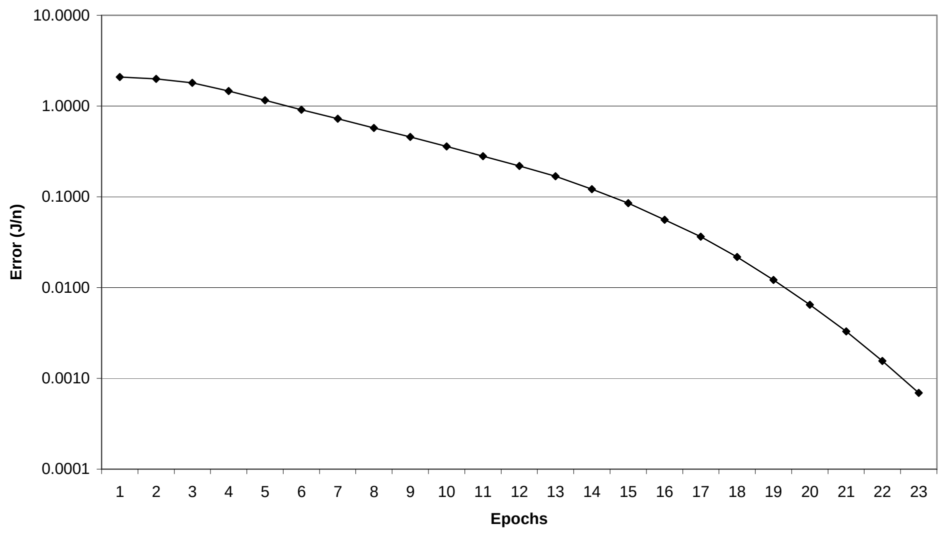

Figure 4.8: Learning curve for neural network

well as indexing and clustering.

When a gesture is time-sampled with a constant sampling period, a sequence of feature vectors is obtained. Each feature vector consists of the x-, y- and z-axis acceleration values, i.e. $a(x, y, z)$. When comparing two gesture sequences using dynamic time warping, one may express the sequence of feature vectors as:

$$A = a_1, a_2, \ldots, a_i, \ldots, a_I \tag{4.2}$$

$$B = b_1, b_2, \ldots, b_j, \ldots, b_J \tag{4.3}$$

where I is the number of feature vectors in the test gesture A, and J is the number of feature vectors in the reference gesture B. The first step in the algorithm is to construct a matrix, D, in which the distances between each feature vector in A and B is tabulated. For each combination of samples in the two gestures, the distance measure used is

$$D(i,j) = ||A_i - B_j|| \tag{4.4}$$

It must now be established which feature vectors in the test gesture correspond to which feature vectors in the reference gesture. Using D one tries to find the correspondence as close to the diagonal of the matrix as possible in order to compute the minimum cost path. One may consider the minimum cost path through the matrix as a sequence of moves. As one cannot move backward in time, the only moves allowed are to the right in the matrix (q), up in the matrix (r), and to the right and up in the matrix (s). We can then calculate the possible moves as

$$q_{i,j} = G_{i-2,j-2} + D_{i-1,j} + Di,j \tag{4.5}$$

$$r_{i,j} = G_{i-1,j-1} + D_{i,j} \tag{4.6}$$

$$s_{i,j} = G_{i-1,j-2} + D_{i,j-1} + Di,j \tag{4.7}$$

where

$$G_{i,j} = \begin{cases} q_{i,j} & q_{i,j} < r_{i,j}, \quad q_{i,j} < s_{i,j} \\ r_{i,j} & r_{i,j} < s_{i,j}, \quad r_{i,j} < q_{i,j} \\ r_{i,j} & q_{i,j} = r_{i,j}, \quad r_{i,j} = s_{i,j} \\ s_{i,j} & s_{i,j} < r_{i,j}, \quad s_{i,j} < q_{i,j} \end{cases} \tag{4.8}$$

One may then use $G_{I,J}$ as a similarity measure. The G matrix indicates how much the gestures differ when best aligned. It is not necessary to perform a full DTW, where the minimum cost path has to be determined by selecting the best path from G.

After calculating $G_{I,J}$ by comparing the test gesture to each of the reference gestures, the minimum value of $G_{I,J}$ is selected to determine which reference gesture resembles the test gesture the most:

$$G = \underset{x}{\operatorname{argmin}} \, G_{I,J}(x) \tag{4.9}$$

where x represents the reference gestures.

In figure 4.9 the G matrix was plotted after using the algorithm to compare a `Circle` test gesture with a `Circle` reference gesture. The $G_{i,j}$ values increase in the direction of the diagonal, due to the spatial and temporal differences between the test gesture and the reference gesture. A deep trough forms in the middle of the graph, indicating that there is a lot of similarity between the test gesture and reference gesture when aligned in the time domain. This can be compared to figure 4.10 where a `Circle` test gesture was compared to a `Square` reference gesture. Once again the $G_{i,j}$ values increase in the direction of the diagonal, but this time the trough is not as pronounced as in the previous example. The final $G_{I,J}$ value is also much larger than in the previous comparison.

In figure 4.11 the G matrix for a `Circle` test gesture and a `Circle` reference gesture was plotted on a 3-dimensional grid to improve the clarity of what is indicated in figure 4.9. The same was done for a `Circle` test gesture and `Square` reference gesture in figure 4.12. As can be seen from the figures, the `Circle` vs. `Circle` grid is flattened out, indicating similarity, but in the `Circle` vs. `Square` grid the differences create a trough and hill. The final value obtained from figure 4.12 is also much larger than the value obtained from figure 4.11, indicating a larger dissimilarity value.

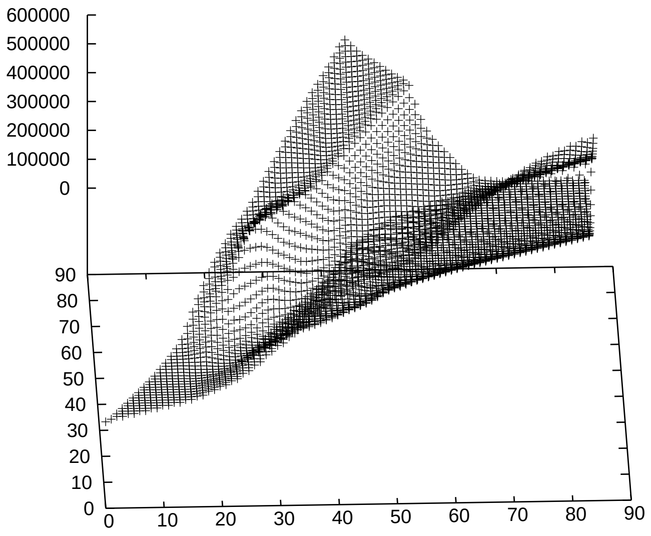

Figure 4.9: Plotting the G matrix: Circle test gesture vs. Circle reference gesture

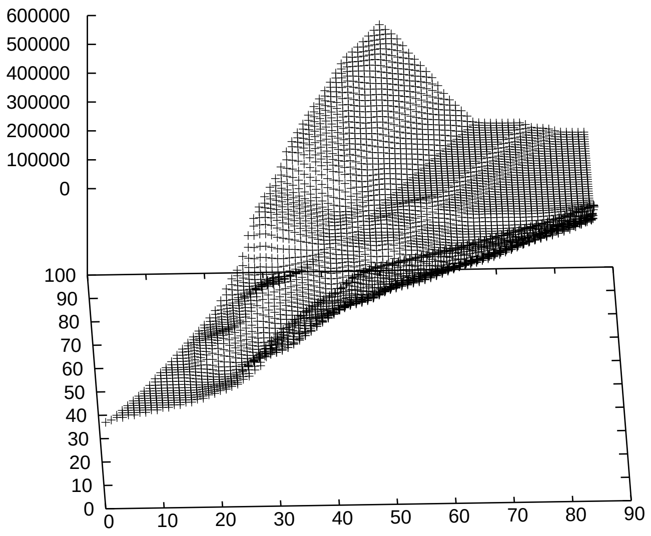

Figure 4.10: Plotting the G matrix: Circle test vs. Square reference

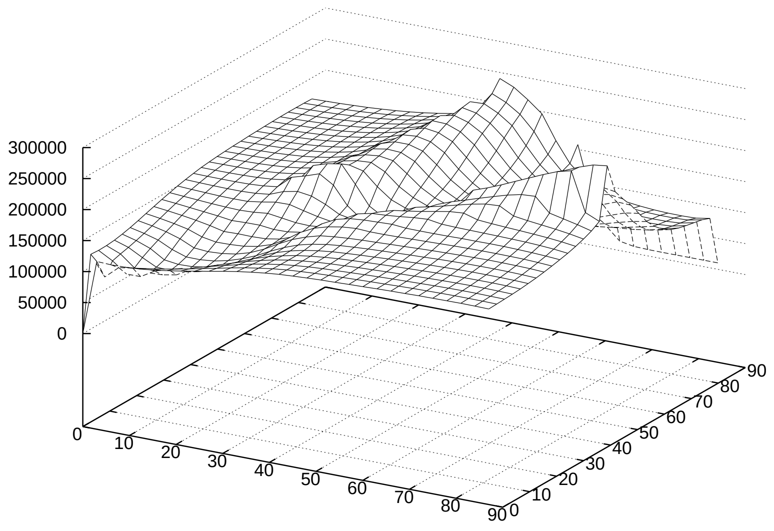

Figure 4.11: Plotting the G matrix on a grid: Circle test gesture vs. Circle reference gesture

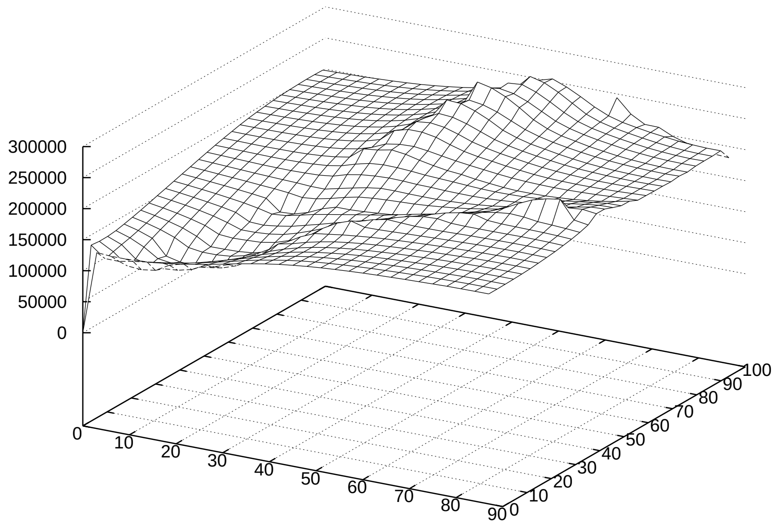

Figure 4.12: Plotting the G matrix on a grid: Circle test gesture vs. Square reference gesture

Chapter 5

Optimizing gesture recognition algorithms

During optimization, algorithm-level optimization must be explored, as well as low-level optimization methods, such as loop unrolling, unrestricted pointer, and custom operations [20]. Performance evaluation includes determining function execution times of the algorithm. This can be specified by giving the execution time (in milliseconds) of each function, as well as the percentage of total execution time of the algorithm.

Algorithm-level optimization replaces time-consuming code segments with code that is more efficient. Timing information must be collected for each function. At this stage, functions that consume the most time can be identified and replaced without optimization at a low-level. For this study, the different algorithms were evaluated and the most efficient algorithm implemented before starting on low-level optimization.

During the low-level optimization process, various methods can be used to reduce branch instructions or increase the basic block size. The basic block size is the ratio of the total number of executed instructions over the total number of dynamic executed branch instructions [20]. Loop fusion and loop unrolling can be used to increase the basic block size. Loop fusion merges basic blocks in the code domain as different code segments are merged, while loop unrolling merges basic blocks in the time domain as different loop iterations are merged. This increases the code size for each loop trip.

Two factors need to be considered when implementing a gesture recognition algorithm on an embedded device:

- Computational efficiency - Some techniques are more computationally efficient than others, and this dictates where the techniques can be used. A technique with low computational efficiency during the learning phase may still be implemented on an embedded device if the computational efficiency of the recognition phase is high. The learning phase may be performed off-line on a computer, and the recognition phase may be implemented on the device.
- Storage efficiency - Different techniques use different representations for their models and require different amounts of storage [33]. This includes both the amount of memory that the model requires and the amount of data required to store the model. Some techniques cannot be implemented on an embedded system due to their large memory requirements.

5.1 Floating point conversion

To reduce chip size and power dissipation, floating-point units are not provided on most embedded processors. Software emulation of floating-point arithmetic is usually implemented, but this degrades the performance of most pattern recognition algorithms. It is possible, however, to write a fixed point implementation of a pattern recognition algorithm [37]. Each floating point variable and constant can be replaced with a 32-bit integer (primitive type "int" in C), where the data representation of the fixed-point variable consists of three fields: sign, integer and fraction. The number of bits assigned to the integer field (the integer word-length, or IWL) determines the value range. The number of bits assigned to the fraction field (the fractional word-length, or FWL) determines the quantization step.

For example, if one selects the FWL as 12 bits, the IWL will be 19 bits (12 + 19 + 1 sign bit = 32 bits). The floating point number 1.875 in decimal (1.111 in binary) may be represented as the fixed point number 7680 in decimal (1111000000000 in binary). Both fixed and variable FWL can be used.

5.2 Optimization considerations

Nissen [38] considered the following optimization techniques while working on the Fast Artificial Neural Network library (FANN) that is included in the Debian Linux distribution:

- Algorithmic optimization - optimizing or changing the algorithms used in the software
- Architectural optimization - modifying the data structures that the algorithm uses
- Cache optimization - modifying the data architecture for sequential data access to improve cache performance
- Common subexpression elimination - manually removing common subexpressions by calculating expressions once and storing the results in local variables
- In-lining of code - avoiding unnecessary overhead for function calls by writing the code directly in the algorithm, or using in-line functions or macros
- Loop unrolling - manually unrolling loops to improve performance, which the compiler may not always perform due to aliasing
- Table lookup - manually calculating values and storing them in a lookup table, instead of calculating them when they are required

With algorithmic optimization, one may try to improve the computational efficiency of the algorithm itself, or make use of different techniques within the algorithm. This is done in neural networks, for example, by making use of the RPROP algorithm instead of the backpropagation algorithm when training the neural network. In some cases it is better to make use of the simpler, more effective algorithm and optimize the actual calculations done in the algorithm.

Architectural optimization consists of modifying or changing the data structures that the algorithm uses. Choosing the optimal data structures allows one to easily loop through the data and optimize the instructions on code level. If the correct data structure is used, the data will be situated closer to each other in memory, allowing for quick access to the data.

Cache optimization is not considered important for embedded devices running a general purpose operating system like Symbian, due to the limited size of the cache typically implemented on such an embedded device. In such a case, the operating system occupies most of the cache and the cache is not suitable to storing large arrays of data. When optimizing for a regular desktop environment or a dedicated embedded device such as a digital signal processor, careful attention must be paid to how data is read from and written to the cache. Data should be stored sequentially, as this improves the performance of the cache [38].

Compilers do perform automatic common subexpression elimination, but it may still be more computationally efficient if done by hand. The results of subexpressions are stored in a lookup table and then read when required. This is especially true of subexpressions inside a loop, in which the calculation can be performed once and then stored. The central loops in the algorithm should be hand optimized, while the more complex loops and not-so-often used loops can be optimized by the compiler.

Inline functions outperform ordinary functions by eliminating the overhead of function calls. This includes tasks such as stack-frame setup, parameter passing, stack-frame restoration and the returning sequence. Besides these key advantages, inline functions also provide the compiler with the ability to perform improved code optimizations [39]. Inline functions are subject to additional optimizations what would not otherwise be possible, because most compilers do not perform interprocedural optimizations. Inline functions work well for small functions (typically three lines or less) or small functions that are called repeatedly. While code expansion can improve speed by eliminating function overhead and allowing for interprocedural compiler optimizations, this is all done at the expense of code size.

Loop unrolling is another optimization technique that may be performed better by hand. This technique improves computational efficiency by preventing the program from jumping back and forth inside a loop and rather let it run instructions sequentially. This also increases speed at the expense of increased program size. When using a resource-constrained device, it may be more computationally efficient to store the results of calculations in a lookup table, instead of calculating them each time they are required.

It is sometimes argued that performance engineering is not a real science, because new algorithms are not created. Nissen [38] does not share this opinion, and gives an example of the importance of performance engineering in a neural network. On an AMD Athlon machine, the *jneural* neural network library uses 509.138 nanoseconds per connection in a fully connected ANN with four layers and 128 neurons in each. On an HP iPAQ the fixed point FANN library uses 188.134 nanoseconds per connection on a similar ANN. This is 2.7 times faster than the *jneural* library. It may be possible to buy a computer which will execute the *jneural* library just as fast as the fixed point library on the iPAQ, but it would not be possible to fit this computer into your pocket.

5.3 Algorithmic optimization

5.3.1 Hidden Markov models

When using the forward-backward algorithm for the evaluation problem, the number of multiplications required in the forward pass is $s^2(T-1)+sT$, and in the backward pass is $(s^2+s)(T-1)$, where s denotes the number of states, and T denotes the total number of elements. The total number of multiplications is therefore equal to $2s^2(T-1)+s(2T-1)$ [34]. If the Viterbi algorithm is used instead, the number of computations may be reduced to $s^2(T-1)+sT$.

By taking logarithms of the model parameters, the Viterbi algorithm can be implemented without any multiplications [41]. The calculations required for this alternative implementation is in the order of N^2T additions, as well as the calculations for preprocessing. As the preprocessing needs to be performed once and saved, its cost is negligible for most systems. We note that when using the Viterbi algorithm to give the maximum likelihood state sequence, no scaling is required if we use the logarithms as discussed in the alternative Viterbi implementation.

The probability computation step is generally performed using the Viterbi algorithm (i.e., the maximum likelihood path is used) and requires on the order of VN^2T computations. For modest vocabulary sizes, e.g., V = 100 words, with an N = 5 state model, and T = 40 observations of the unknown word, a total of 10^5 computations is required for recognition (where each computation is a multiply operation, an add operation and a calculation of the observation density, $\mathbf{b}(0)$).

5.3.2 Neural networks

In a feedforward ANN an input can easily be propagated through the network and evaluated to an output. It is more difficult to compute a clear output from a network where connections are allowed in all directions (as in the brain), as this will create loops. There are ways of dealing with these loops in recurrent networks, but feedforward networks are usually a better choice for problems that are not time dependent.

The running time of the execution of a neural network (not the training time) can be estimated as follows:

$$T = cP + (n - n_i)G \tag{5.1}$$

where c is the number of connections, n is the total number of neurons, n_i is the number of input and bias neurons, P is the cost of multiplying the weight with the input and adding it to the sum, G is the cost of the activation function and T is the total cost. If the ANN is fully connected, l is the number of layers and n_l is the number of neurons in each layer (not counting the bias neuron), this equation can be rewritten as

$$T = (l - 1)(n_l^2 + n_l)P + (l - 1)n_l G \tag{5.2}$$

This equation shows that the total cost is dominated by P in a fully connected ANN. This means that if one wants to optimize the execution of a fully connected ANN, one needs to optimize P and the retrieval of the information needed to compute P. While training an MLP is processor intensive, computing the output of a trained MLP requires very little processor time, particularly since all internal quantities can be modelled using integers and nonlinear functions replaced by a look-up table [39].

5.4 Performance measurement

A sampling profiler operates by frequently sampling the position of the instruction pointer while the program runs [40]. This generates a huge amount of raw data which is then processed to generate profiling data. Sampling profilers can often tell the programmer exactly in which line of code most of the time is spent. Commercial profilers usually work this way because modifications to the application code is unnecessary. Another profiling method that is commonly used is to explicitly time blocks of code. These measurements can then be displayed in real-time and aid in finding transient performance problems.

5.5 Other optimization techniques

The Symbian OS Performance Tips booklet [43] gives the following tips for further optimization:

- Try to create data structures only once and re-use them in loops.
- Remove variables in loops that will stay the same (this can lead to inefficient heap usage.
- Check if unnecessary variable casting is done (also known as type coercion).
- Check for inefficient file usage.
- Remove function calls in a loop's condition statement.
- It is unnecessary to unroll loops. With modern compilers this type of optimization is no longer necessary, and may even be counterproductive. The compiler may perform this optimization where appropriate.
- If the *const* qualifier is used appropriately to mark read-only variables, the compiler may generate more efficient code.

Chapter 6

Results

6.1 Comparing the gesture recognition algorithms

6.1.1 Hidden Markov model

The confusion matrices in tables 6.1 and 6.2 were obtained by means of leave-one-out validation testing. The Hidden Markov Model was trained with 79 of the 80 available samples and then tested using the remaining sample. Initially a 4-state HMM was used, but the recognition accuracy of the HMM proved to be too low to be able to accurately compare the model with the other algorithms. Using 8 states for the HMM increases the recognition accuracy to above 90%, which is approximately equal to the accuracy obtained with the other algorithms.

For the HMM with 4 states, a total of 53 of the 80 samples were correctly classified, for a total accuracy of 66.25%. For the HMM with 8 states, a total of 77 of the 80 samples were correctly classified, for a total accuracy of 96.25%. It is interesting to note that the HMM with 4 states is unable to distinguish between the circular gestures, incorrectly classifying all *Right Circle* gestures as *Left Circle* gestures.

Table 6.1: Confusion matrix for HMM with 4 states

	Left Circle	Left Square	Left Triangle	Left Infinity	Right Circle	Right Square	Right Triangle	Right Infinity
Left Circle	0.7	0.0	0.0	0.1	0.0	0.0	0.2	0.0
Left Square	0.2	0.5	0.0	0.0	0.0	0.2	0.1	0.0
Left Triangle	0.0	0.0	0.1	0.9	0.0	0.0	0.0	0.0
Left Infinity	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
Right Circle	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Right Square	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
Right Triangle	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Right Infinity	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Table 6.2: Confusion matrix for HMM with 8 states

	Left Circle	Left Square	Left Triangle	Left Infinity	Right Circle	Right Square	Right Triangle	Right Infinity
Left Circle	0.9	0.1	0.0	0.0	0.0	0.0	0.0	0.0
Left Square	0.1	0.9	0.0	0.0	0.0	0.0	0.0	0.0
Left Triangle	0.0	0.0	0.9	0.1	0.0	0.0	0.0	0.0
Left Infinity	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
Right Circle	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
Right Square	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
Right Triangle	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Right Infinity	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

6.1.2 Neural network

The confusion matrix in table 6.3 was obtained by means of leave-one-out validation testing. The neural network was trained with 79 of the 80 available samples and then tested using the remaining sample. This process was repeated for each of the 10 samples per gesture. The rows represent the sample being tested, and the columns indicate as which gesture the sample was recognized.

Table 6.3: Confusion matrix for ANN

	Left Circle	Left Square	Left Triangle	Left Infinity	Right Circle	Right Square	Right Triangle	Right Infinity
Left Circle	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Left Square	0.0	0.6	0.4	0.0	0.0	0.0	0.0	0.0
Left Triangle	0.0	0.0	0.9	0.1	0.0	0.0	0.0	0.0
Left Infinity	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
Right Circle	0.0	0.0	0.1	0.0	0.9	0.0	0.0	0.0
Right Square	0.0	0.0	0.0	0.0	0.0	0.8	0.2	0.0
Right Triangle	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Right Infinity	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

A total of 72 of the 80 samples were correctly classified, for an overall accuracy of 90%.

6.1.3 Dynamic time warping

The training and recognition procedures in DTW are potentially much faster than other techniques used for gesture recognition, such as HMMs and ANNs. This was evaluated by implementing a gesture recognition system using the gesture data gathered in section

4.1. Since there are no training steps or preprocessing required for the DTW algorithm, the raw gesture data could be evaluated directly. Leave-one-out validation was used to test the algorithm, by using the DTW algorithm to compare each one of the 8 gestures to the remaining 79 reference gestures.

Table 6.4: Confusion matrix for DTW

	Left Circle	Left Square	Left Triangle	Left Infinity	Right Circle	Right Square	Right Triangle	Right Infinity
Left Circle	0.8	0.0	0.0	0.0	0.0	0.2	0.0	0.0
Left Square	0.0	0.9	0.0	0.0	0.0	0.1	0.0	0.0
Left Triangle	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
Left Infinity	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
Right Circle	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
Right Square	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
Right Triangle	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Right Infinity	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

With the DTW algorithm, a total of 77 of the 80 samples were correctly classified, for an overall accuracy of 96.25%. This compares very well with the HMM algorithm with 8 states, and the DTW algorithm is therefore considered sufficiently accurate as to implement it.

6.1.4 Measuring execution time

The execution time of the three algorithms was first measured on a HP Pavilion dv9086ea notebook with an Intel Core 2 Duo T7200 processor running at 2 GHz and with 1 GB of RAM. Testing was performed on Ubuntu Linux v. 7.04 Feisty Fawn. Execution time was measured using the UNIX `time` command. To measure the execution time of the algorithm, the following steps were followed:

- All optimization flags were disabled before the compilation of each algorithm.
- Output to `stdout` and `stderr` was suppressed by running the algorithm with the command `./name_of_algorithm >nul 2>&1`
- 1000 program executions of each algorithm were measured, and the average execution time computed.

The reasoning behind measuring the average of 1000 program executions, instead of measuring the program execution time directly, can be explained by ways of the following analogy: If one would like to measure the width of a piece of paper, one would rather measure the width of a stack of 200 pieces of paper instead of buying an expensive microscope to measure the width of one piece of paper. Even though it is possible to measure the program execute time of one iteration of the algorithm, the accuracy of the measurement is improved by measuring the average of 1000 iterations of the algorithm. This is due to the fluctuations in the program execution time when trying to measure only one iteration of the algorithm.

The measured time is the time the algorithm takes to recognize one gesture. In the case of this study, the gesture being tested was the `Left Circle` gesture. The results of performance measurements can be seen in table 6.5 and figures 6.1 and 6.2.

The measurements to determine the required amount of memory include the amount of memory required by the Python interpreter. This is due to the interpreter being used to run multiple iterations of the algorithm, in order to accurately measure the performance. As the required amount of memory for each algorithm does not differ significantly, using required memory as a performance metric is considered irrelevant.

Table 6.5: Comparing algorithm performance characteristics

Characteristic	HMM (4 / 8 states)	ANN	DTW
Recognition performance	Medium	Slow	Fast
Recognition (preprocessing)	2.8 ms	1.8 ms	None required
Recognition (algorithm)	10.5 ms / 12.2 ms	23.02 ms	8.31 ms
Training time	Long / Extensive	Medium	No training
Size of network on disk	4382 bytes / 13740 bytes	591163 bytes	5926 bytes
Memory required	1.5 MB / 1.5 MB	1.4 MB	1.5 MB

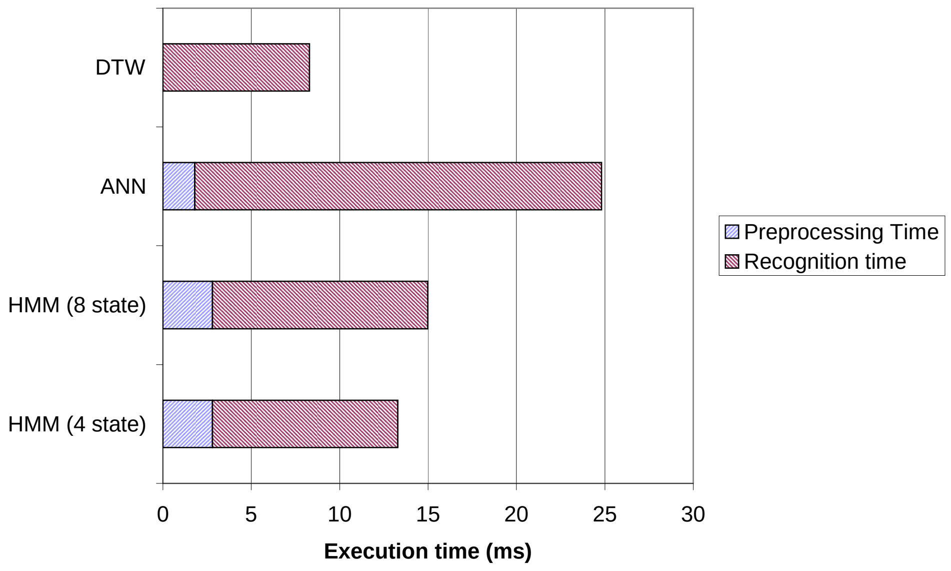

Figure 6.1: Execution times for all the algorithms

The average execution times for the three algorithms is provided in figure 6.1. It is interesting to note that the neural network's execution time is unusually long in comparison to the other two algorithms. This is due to the large dimensions of the neural network - there are 150 input neurons, 108 hidden neurons and 8 output neurons in the network. The size of the neural network may be decreased by decreasing the amount of input neurons required, by first performing feature extraction on the data. It would be interesting to test whether the additional overhead incurred by preprocessing will shorten the execution time of the network significantly, or enough to improve the overall performance.

The DTW algorithm's execution time is slightly faster than that of the 4-state HMM, but the 4-state HMM's recognition accuracy is much lower (66% vs. 96%). The DTW algorithm outperforms the 8-state HMM with regards to execution time, and with similar recognition accuracy. Another advantage of the DTW algorithm is that it requires no preprocessing - the calculations may be performed directly on the raw sampled data. Based on these results the decision was made to implement and optimize the DTW algorithm on the mobile device.

The neural network requires a large amount of storage space (577 KB) compared to the HMM and DTW algorithms (13KB and 6 KB respectively). Note that the graph in figure 6.2 is plotted on a logarithmic scale in order to observe the actual values of the HMM and DTW algorithms. This is due to the large size of the neural network. The DTW algorithm outperforms the ANN and HMM algorithms with regards to the required storage space. It should be noted, however, that as the amount of gestures to be recognized increase, the storage space required by the DTW algorithm will increase linearly. This is not the case with the ANN and DTW algorithms, as the sizes of the networks will remain static, unless they have to be enlarged to increase recognition accuracy.

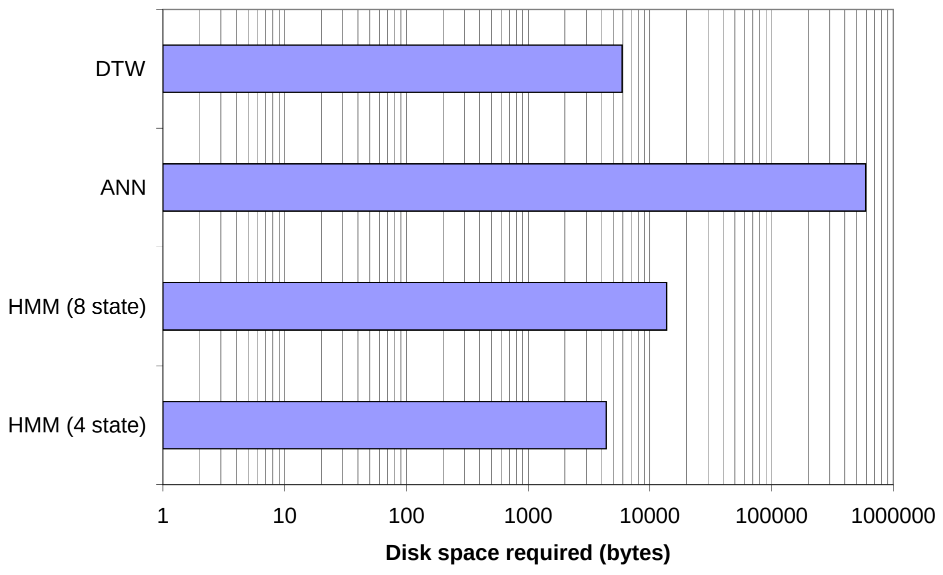

Figure 6.2: Storage space required for storing the trained networks

6.2 Profiling the gesture recognition algorithm on the mobile device

It does not make sense to use a profiler tool on the algorithm implemented on the mobile device, as the algorithm is contained in only a few subroutines. Profilers work well in cases where there are many subroutines and the source of a bottleneck must be determined. Therefore only a timer routine was used to profile the gesture recognition algorithm. When the algorithm on the mobile device was profiled, the following steps were followed:

- Ensure that no other programs are running at the same time.
- Measure program start time and measure program end time with microsecond resolution.
- Subtract start time from end time to get running time.

- Initially the running time will get shorter after each iteration. Record 5 running times to calculate an average running time when successive running times differ by less than 10ms.

To accurately determine the execution time of the algorithm on the mobile device, the Symbian `TTime::HomeTime()` function was used. This timer has a microsecond resolution.

6.3 Porting the gesture recognition algorithm to Symbian OS

The gesture recognition algorithm was ported to the mobile device by making use of Nokia's Open C platform [55]. Open C is a set of POSIX libraries to enable standard C programming on Symbian Series 60 devices. Open C implements a version of the `libc` standard C library that includes standard input/output routines, database routines, bit operators, string operators, character tests and character operators, storage allocation, time functions and signal handling. The Open C version used in this study was 3.0.1. The development environment used was Nokia's Carbide.c++ v.1.2 IDE, that is based on the open source Eclipse development tools [58].

Due to the limited amount of memory available on the mobile device, the original DTW algorithm did not run as expected. This was attributed to allocated memory not being freed after each iteration of the DTW algorithm. On a PC, this is not a problem, as there is sufficient memory available to hold all the required data structures for each iteration of the DTW algorithm and then freeing the memory on exit. The program was modified to free the allocated memory after each iteration of the DTW algorithm.

6.4 Optimizing the gesture recognition algorithm on the mobile device

Modern compilers perform many optimizations (e.g. loop unrolling) automatically. Before manually performing low-level optimizations, the optimization flags on the compiler were adjusted to see whether performance could be improved. The compiler that was used to compile code for the mobile device is the GCCE compiler distributed as part of the Nokia S60 3rd Edition SDK for Symbian OS Feature Pack 1 (FP1).

To modify optimization flags in the GCCE compiler, the `gcce.mk` file in the

`C:\Symbian\9.2\S60_3rd_FP1\Epoc32\tools\compilation_config`

directory may be edited and the `REL_OPTIMIZATION` field modified for the phone release build configuration. The GCCE compiler supports 3 levels of optimization with additional optimization flags. On optimization level 1, around 18 different optimizations are performed, some of which are shown in table 6.6. On optimization level 2, an additional 21 optimizations are performed. On optimization level 3, an additional 3 optimizations are performed:

- `-finline-functions` - Integrates all simple functions into their callers.
- `-funswitch-loops` - Move branches with loop invariant conditions out of the loop.
- `-fgcse-after-reload` - A redundant load elimination pass is performed after reload.

The results of the different compiler optimization flags, together with the results of the other optimizations described in this section, is provided in figure 6.3. It should be noted that the optimizations as shown in the figure were performed incrementally, e.g. the optimization where the loading of the test gesture was moved out of the loop includes the optimization where only integer arithmetic was used.

Table 6.6: Optimization flags enabled on optimization level 2

Optimization flag	Description
`-fdefer-pop`	Always pop the arguments to each function call as soon as that function returns.
`-fdelayed-branch`	Attempt to reorder instructions to exploit instruction slots.
`-fguess-branch-probability`	Use heuristics to guess branch probabilities.
`-fcprop-registers`	A copy-propagation pass is performed to try to reduce scheduling dependencies.
`-floop-optimize`	Moves constant expressions out of loops and simplify exit test conditions.
`-fif-conversion`	Attempt to transform conditional jumps into branch-less equivalents.
`-fif-conversion2`	Use conditional execution to transform conditional jumps into branch-less equivalents.
`-ftree-ccp`	Perform sparse conditional constant propagation (CCP) on trees.
`-ftree-dce`	Perform dead code elimination (DCE) on trees.
`-ftree-dominator-opts`	Perform a variety of simple scalar cleanups e.g constant/copy propagation
`-fmerge-constants`	Attempt to merge identical constants across compilation units.

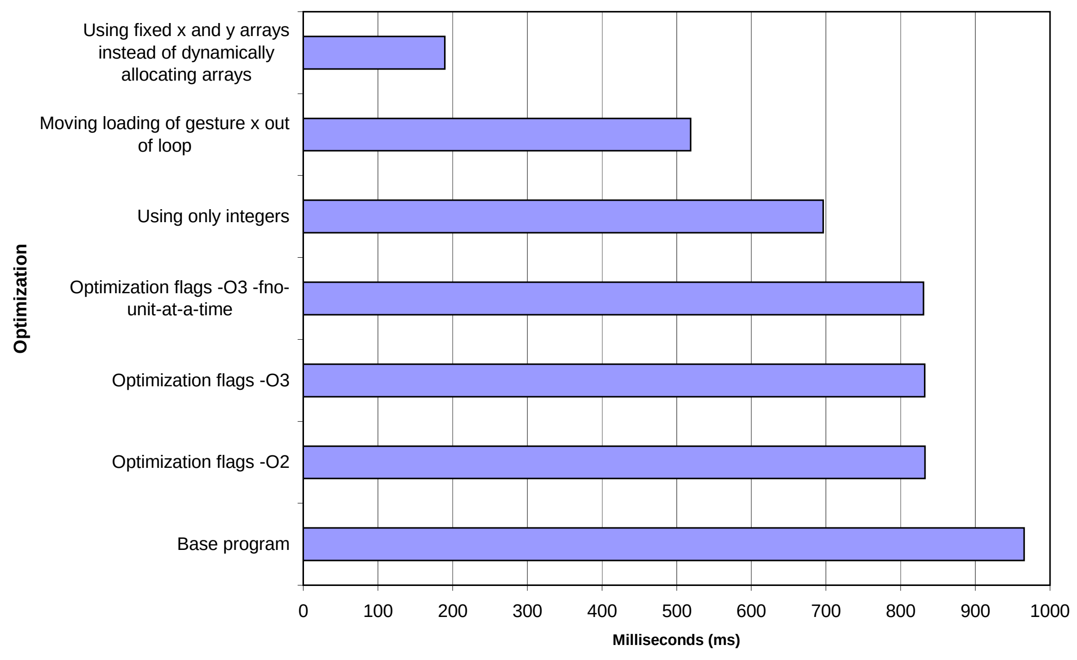

Figure 6.3: Execution times after performing different optimizations

The flag `-fno-unit-at-a-time` prevents the compiler from parsing the whole compilation unit before starting to produce code. Parsing the whole compilation unit allows some extra optimizations to take place but consumes more memory (in general).

Having the compiler perform automatic optimization reduces the execution time by about 14%. There is not much difference in improvement between the different optimization levels, due to the simplistic nature of the algorithm. The main algorithm is contained in one function, so inlining functions does not really make a difference.

After automatic compiler optimization has been performed, any unnecessary variables and function calls in all loops' condition statements, as well as in the rest of the program, were replaced with constants. This did not prove to have any effect on the execution time.

By converting the floating point variables in the program to integer variables, a 16% reduction in execution time was achieved. To reduce chip size and power dissipation, floating-point units are not provided on most embedded processors. Software emulation of floating-point arithmetic is usually implemented. However, the mobile device that these optimizations were performed on, the Nokia N95, does include a Floating-Point Unit (FPU). This means that for mobile devices without a FPU, the performance gain will be even higher.

Architectural optimization was also carried out. In the base program (without any optimization) the gesture file being recognized was loaded each time the DTW algorithm was performed. By only loading the file once at the start of the program and then using the same data structure for future calculations, a further 26% reduction in execution time was achieved. This indicates that by properly refactoring a section of code, large performance gains are achievable.

The base program dynamically allocates two-dimensional arrays after the gesture files are loaded. By replacing the dynamically allocated array with a fixed array, an additional 64% reduction in execution time was achieved. This is quite significant, and it indicates the very large amount of overhead that is required to dynamically allocate arrays using the `malloc()` function.

6.5 Implementation of a user application utilizing gesture recognition on a mobile device

A user application was implemented to test the real-world functionality of the gesture recognition algorithm. The user application was developed in the Python programming language and executed on the mobile device using Nokia's Python for Series 60 (S60) version 1.4.1 [56] utilities. Using Python allows one to rapidly prototype a graphical user interface (GUI) and other functionality by making use of the built-in APIs to provide, for example, sound and graphics capabilities. An example of the user application running on the mobile device is shown in figure 6.4.

To have the system learn a new gesture, the user can select the `New Gesture` command from the pop-up menu. When the user starts the moving the device, the application records the gesture until the device stops moving. The recorded gesture is then stored as a reference gesture on the device. To recognize a gesture, the user selects the `Recognize` command from the pop-up menu. The application records the test gesture as soon as the user starts moving the device. When the device stops moving, the application executes the gesture recognition algorithm and displays the recognized gesture as a graphic on the screen. The recognized gesture is also spoken out loud using the text-to-speech functionality of the Nokia Python Audio API.

A sliding window algorithm, similar to the one suggested by [2], was used to detect the

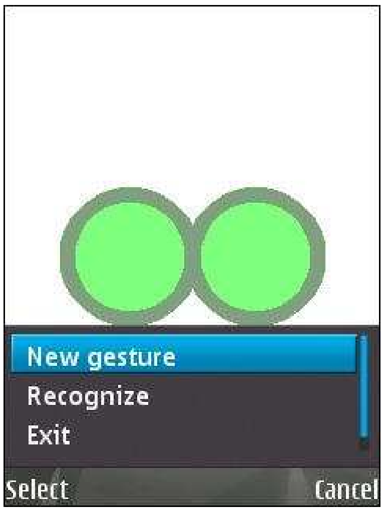

Figure 6.4: User application running on the mobile device

beginning and end of the gestures. The algorithm in [2] suggests that ranges where the variance is greater than a set threshold may be considered to be periods of activity. The window size and threshold value may be found analytically, based on the sensor noise floor and the minimum attack speed considered to represent a deliberate motion. For the user application the start of a gesture was recorded when the variance was greater than a set threshold. The end of the gesture was recorded when the variance was lower than a set threshold, based on a sliding window where the variance was calculated using a window of previous samples.

The gesture recognition algorithm (written in C) was linked into the Python program as a dynamically linked library (DLL). Wrapper code was created for the C algorithm in order to link it into the Python program.

The user application was converted into a standalone Python program on the Symbian device through the Ensymble developer utilities for Symbian S60 [57]. It can also run as a script in the Python for S60 shell.

When using the Symbian for 3rd edition SDK FP1, the program has to be signed with a cryptographical signature and a digital certificate must be generated. This may be done by making use of the Symbian `createsis` command. From Symbian 3rd Edition onwards, it is not possible to install an application on a Symbian device without signing the application first.

Chapter 7

Conclusion

7.1 Summary of the work

This research provides two contributions to the field of gesture recognition. It was shown that gesture recognition can be performed on a mobile device with high accuracy for a reasonable amount of gestures, without the need for extensive preprocessing. It was also shown that partial dynamic time warping is a very effective technique to recognize accelerometer-based gestures, with very low computational overhead and minimal training required.

The applicability of optimization techniques to resource-constrained devices was also investigated, with some interesting findings. Dynamically allocating arrays on a mobile device proved to be one of the largest contributing factors to computational overhead on a resource-constrained device. It is possible to optimize even a simple algorithm by refactoring the code for speed and performance.

A HMM with 4 states was able to correctly classify a total of 53 of the 80 samples, for a total accuracy of 66.25%. As the recognition accuracy of the HMM proved to be too low to be able to accurately compare the model with the other algorithms, a HMM with 8 states was used. For the HMM with 8 states, a total of 77 of the 80 samples were correctly classified, for a total accuracy of 96.25%.

Regarding the ANN, a total of 72 of the 80 samples were correctly classified, for a total accuracy of 90%. Regarding the DTW algorithm, a total of 77 of the 80 samples were correctly classified, for a total accuracy of 96.25%. This compares very well with the HMM algorithm with 8 states, and the accuracy of the DTW algorithm was considered sufficient for implementation.

The ANN had the slowest computational performance, due to the large size of the neural network. The HMM performed better, but the DTW algorithm proved to be the fastest with comparable recognition accuracy. The DTW algorithm has an average recognition time of 8.31ms, compared to the HMM algorithm at 15ms, and the ANN algorithm at 24.82ms. Another advantage of the DTW algorithm is that there is no preprocessing required whatsoever - the calculations can be performed directly on the raw sampled data.

The model was empirically validated by implementing it in a user application on the mobile device. After the gesture recognition algorithm was implemented on the mobile device, it was discovered that the system does have some problems handling dynamical differences (intensive vs. phlegmatic), but performs well with regards to spatio-temporal differences.

7.2 Critical evaluation of own work

Partial dynamic time warping proves to be effective for recognizing accelerometer-based gestures. It has very low computational overhead and has low memory requirements for a reasonable amount of gestures. It does not require extensive training, as is the case with both hidden Markov models and artificial neural networks.

A well-defined and sufficiently constrained recognition problem (small intraclass variations and large interclass variations) will lead to a compact pattern recognition representation and a simple decision making strategy. It is therefore of importance that the gestures to be recognized are sufficiently different from one another in order to create large interclass variations.

The reasoning behind using only 8 different gestures was explained earlier in the text, but the sample size used per gesture may be a cause for concern. In order to fully test the validity of the partial dynamic time warping algorithm used, the amount of samples per gesture should be increased. This is left as future work, as the objective of this study was only to evaluate the various algorithms and optimize the most efficient one.

The algorithm implemented is not a full implementation of the dynamic time warping algorithm. A minimum cost path obtained from the distance matrix is used as a similarity measure, without having to calculate the final time warped signal. To the author's knowledge, this method has not yet been applied to accelerometer-based gesture recognition.

7.3 Future work

Future work includes performing user testing to gauge recognition accuracy when different users are involved. Both user-dependent and user-independent testing should be performed. Informal testing shows good user-dependent accuracy, but suboptimal user-independent accuracy. The algorithm simplifies training by only requiring one gesture sample to be performed. This means that the system can be retrained for different users very easily, making user-independent accuracy unnecessary.

The size of the neural network can be decreased by decreasing the amount of input neurons required. This can be done by performing feature extraction on the data. It would be interesting to see if the additional overhead incurred by preprocessing will decrease the execution time of the network significantly, or enough to improve the overall performance.

The applicability of the partial dynamic time warping algorithm to other pattern recognition problems should be explored. It is possible that the similarity measure may be used as a computationally efficient way to solve pattern recognition problems in other domains where there are a limited amount of patterns, but a large amount of spatio-temporal variability.

The algorithm may be ported to other resource-constrained devices, such as wireless sensor nodes. As the algorithm was implemented in the C programming language and was optimized with regards to stack and heap size, porting it to other microcontroller architectures should prove to be unproblematic.

By utilizing accelerometer-based gesture recognition techniques and optimizing the gesture recognition algorithms for resource-constrained devices, truly mobile applications in the field of Human-Computer Interaction (HCI) can be created by performing gesture recognition on-board a mobile device. Users will be able to manipulate objects on the mobile device directly using hand gestures, or use gestures to issue specific commands.

Bibliography

[1] T. Baudel and M. Beaudoin-Lafon, "Charade: remote control of objects using free-hand gestures," *Communications of the ACM*, vol. 36, no. 7, pp. 28-35, July 1993.

[2] A.Y. Benbasat and J.A. Paradiso, "An Inertial Measurement Framework for Gesture Recognition and Applications," *Lecture Notes in Computer Science: Gesture and Sign Language in Human-Computer Interaction*, vol. 2298, no.2002, pp. 77-90, 2002.

[3] T. Westeyn et al., "Georgia tech gesture toolkit: supporting experiments in gesture recognition," *Proceedings of the 5th International Conference on Multimodal interfaces, Vancouver, British Columbia, Canada*, pp. 85-92, 5-7 November 2003.

[4] P. Keir et al., "Gesture-recognition with Non-referenced Tracking," *Proceedings of the IEEE conference on Virtual Reality, Alexandria, Virginia, USA*, pp. 137, 25-29 March 2006.

[5] E. Farella et al., "Design and implementation of WiMoCA Node for a body area wireless sensor network," *Proceedings of the IEEE Conference on Sensor Networks, Montreal, Canada,* pp. 342-347, 14-17 August 2005.

[6] D. Geer, "Will gesture recognition technology point the way?," *IEEE Computer*, vol. 37, no. 10, pp. 20-23, 2004.

[7] A. Feldman, E.M. Tapia, S. Sadi, P. Maes, C. Schmandt, “ReachMedia: On-the-move interaction with everyday objects,” *Ninth IEEE International Symposium on Wearable Computers (ISWC’05), Osaka, Japan*, pp. 52-59, 18-21 October 2005.

[8] W. Chou, *Pattern recognition in speech and language processing,* CRC Press, Boca Raton, Florida, 2003.

[9] A. Wilson and A. Bobick, “Parametric hidden Markov models for gesture recognition,” *Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 9, pp. 884-900, 1999.

[10] H. Lee and J. Kim, “An HMM-based threshold model approach for gesture recognition,” *Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 10, pp. 961-973, 1999.

[11] A. Buys, *Research guide for post-graduate students*, Department of Engineering and Technology Management, University of Pretoria, p. 7, Feb. 2006.

[12] K. Wadleigh, *Software optimization for high-performance computing,* Prentice Hall, Upper Saddle River, New Jersey, 2000.

[13] H. Muller, C. Randell, and A. Moss, “A 10mW Wearable Positioning System,” *Proceedings of the 10th IEEE International Symposium on Wearable Computers, Montreux, Switzerland*, pp. 47-50, 11-14 October 2006.

[14] J. Mouton, *How to succeed in your master’s and doctoral studies: a South African guide and resource book*, Pretoria: Van Schaik, pp. 163-164, 2001.

[15] A. Jain, R. Duin, and J. Mao, “Statistical pattern recognition: a review,” *Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 4-37, 2000.

[16] D. Lowe and A. Webb, "Optimized feature extraction and the Bayes decision in feed-forward classifier networks," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 355-364, 1991.

[17] E. Choi et al., "Beatbox music phone: gesture-based interactive mobile phone using a tri-axis accelerometer," *IEEE International Conference on Industrial Technology (ICIT 2005), Hong Kong,* pp. 97-102, 15-18 December 2005.

[18] K. Van Laerhoven et al., "Towards a wearable inertial sensor network," *IET Seminar Digests*, vol. 2003, no. 10350, pp. 125-130, 2003.

[19] K . Van Laerhoven and H. Gellersen, "Spine versus porcupine: a study in distributed wearable activity recognition," *Eighth International Symposium on Wearable Computers (ISWC 2004), Arlington, VA, USA*, vol. 1, pp. 142-149, 31 October - 3 November 2004.

[20] I. Ozer, T. Lu, and W. Wolf, "Design of a real-time gesture recognition system: high performance through algorithms and software," *IEEE Signal Processing Magazine*, vol. 22, no. 3, pp. 57-64, 2005.

[21] J. Kumagai, "Talk to the machine," *IEEE Spectrum*, vol. 39, no. 9, pp. 60-64, 2002.

[22] H. Brashear et al., "Using multiple sensors for mobile sign language recognition," *Proceedings of the Seventh IEEE International Symposium on Wearable Computers, Boston, MA, USA*, pp. 45-52, 11-13 October 2003.

[23] J. Mäntyjärvi, J. Himberg, and T. Seppanen, "Recognizing human motion with multiple acceleration sensors," *IEEE International Conference on Systems, Man, and Cybernetics, Arizona, USA*, pp. 747-752, 7-10 October 2001.

[24] E. Barnard and D. Casasent, "Invariance and neural nets," *IEEE Transactions on Neural Networks*, vol. 2, no. 5, pp. 498-508, 1991.

[25] T. Pylvänäinen, "Accelerometer Based Gesture Recognition Using Continuous HMMs," *Lecture Notes in Computer Science: Pattern Recognition and Image Analysis*, vol. 3522, no. 2005, 2005.

[26] V. Mäntylä et al., "Hand gesture recognition of a mobile device user," *IEEE International Conference on Multimedia and Expo, New York, NY, USA*, vol. 1, pp. 281-284, 30 July - 2 August 2000.

[27] G. Chambers et al., "Hierarchical recognition of intentional human gestures for sports video annotation," *Proceedings of the 16th International Conference on Pattern Recognition, Quebec City*, pp. 1082-1085 vol.2, 11-15 August 2002.

[28] T. Westeyn et al., "Recognizing mimicked autistic self-stimulatory behaviors using HMMs," *Proceedings of the Ninth IEEE International Symposium on Wearable Computers, Osaka, Japan*, pp. 164-167, 18-21 October 2005.

[29] G. Bailador, D. Roggen, G. Tröster, and G. Triviño, "Real time gesture recognition using Continuous Time Recurrent Neural Networks," *Proceedings of the 2nd International Conference on Body Area Networks (BodyNets), Florence, Italy,* June 2007.

[30] Y.H. Hu and J. Hwang, *Handbook of neural network signal processing*, CRC Press, Boca Raton, Florida, 2002.

[31] S.T. Welstead, *Neural network and fuzzy logic applications in C/C++*, John Wiley & Sons, New York, 1994.

[32] E. Charniak, "Bayesian networks without tears", *AI Magazine*, vol. 12, no. 4, pp. 50-63, 1991.

[33] S. Rabin, *AI game programming wisdom 3*, Charles River Media, Boston, Massachusetts, 2006.

[34] L.F.C. Pessoa, "Multilayer perceptrons versus Hidden Markov Models: Comparisons and applications to image analysis and visual pattern recognition", Ph.D. thesis, Georgia Institute of Technology, 1995.

[35] K. Murakami and H. Taguchi, "Gesture recognition using recurrent neural networks", *Proceedings of the SIGCHI conference on human factors in computing systems, New Orleans, Louisiana, USA*, pp. 237-242, 28 April - 5 June 1991.

[36] D. Xu, "A neural network approach for hand gesture recognition in virtual reality driving system of SPG", *18th International Conference on Pattern Recognition (ICPR'06), Hong Kong*, pp. 519-522, 20-24 August 2006.

[37] Y.S. Moon, C.C. Leung and K.H. Pun, "Fixed-point GMM-based speaker verification over mobile embedded system", *Proceedings of the 2003 ACM SIGMM workshop on Biometrics methods and applications, Berkeley, CA, USA*, pp. 53-57, 2-8 November 2003.

[38] S. Nissen, "Implementation of a fast neural network library (FANN)", Graduate Report, University of Copenhagen (DIKU), 2003.

[39] M. De Loura, *Game programming gems 2*, Charles River Media, Boston, Massachusetts, 2001.

[40] D. Treglia, *Game programming gems 3*, Charles River Media, Boston, Massachusetts, 2002.

[41] L.R. Rabiner and B.H. Juang, "Fundamentals of speech recognition", Prentice Hall, Upper Saddle River, New Jersey, 1993.

[42] Cambridge University Engineering Department, "HTK Speech Recognition Toolkit", 2008, http://htk.eng.cam.ac.uk/. Last accessed on 14 April 2008.

[43] A. Langstaff, "Essential Symbian OS Performance Tips", 2008, http://developer.symbian.com/main/learning/press/books/pdf/Performance_Tips.pdf. accessed on 14 April 2008.

[44] Free Software Foundation, GCC Online Documentation, 2008, http://gcc.gnu.org/onlinedocs/. Last accessed on 14 April 2008.

[45] M.H. Ko et al., "Online context recognition in multisensor systems using dynamic time warping", *Proceedings of the 2005 International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Melbourne, Australia*, pp. 283-288, 5-8 December 2005.

[46] J. Coleman, "Introducing speech and language processing", Cambridge University Press, Cambridge, UK, 2005.

[47] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43-49, 1978.

[48] A. Graps, "An introduction to wavelets", *IEEE Computational Science & Engineering*, vol. 2, no. 2, pp. 50-61, 1995.

[49] J. Heaton, "Introduction to neural networks with Java", Heaton Research Inc., Chesterfield, Missouri, 2005.

[50] S. Nissen, "Fast Artificial Neural Network Library (FANN)", 2008, http://fann.sourceforge.net/. Last accessed on 17 April 2008.

[51] Georgia Institute of Technology Contextual Computing Group, "Georgia Tech Gesture Recognition Toolkit", 2006, http://gt2k.cc.gatech.edu/. Last accessed on 18 April 2008.

[52] V.N. Vapnik, "Statistical learning theory", Wiley & Sons, New York, 1998.

[53] AiLive Inc., “LiveMove White Paper”, 2006, http://www.ailive.net/papers/LiveMoveWhitePaper_en.pdf. Last accessed on 17 April 2008.

[54] A. Wilson and S. Shafter, “XWand: UI for intelligent spaces”, *Proceedings of the SIGCHI Conference on human factors in computing systems, Lauderdale, Florida*, pp.545-552, 5-10 April 2003.

[55] Nokia Research Center, “Open C: Standard-based Libraries for Symbian-based Smartphones”, 2008, http://opensource.nokia.com/projects/openc/. Last accessed on 17 April 2008.

[56] Nokia Research Centre, “Python for S60,” 2008, http://opensource.nokia.com/projects/pythonfors60/. Last accessed on 17 April 2008.

[57] J. Ylänen, “The Ensymble developer utilities for Symbian OS”, 2007, http://www.nbl.fi/ nbl928/ensymble.html. Last accessed on 17 April 2008.

[58] Nokia, “Carbide Development Tools”, 2008, http://www.forum.nokia.com/main/resources/tools_and_sdks/carbide/index.html. Last accessed on 17 April 2008.

[59] J.O. Wobbrock, A.D. Wilson and Y. Li, “Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes,” *Proceedings of the 20th annual ACM symposium on user interface software and technology, Newport, RI, USA,* pp. 159-158, 7-10 October 2007.

[60] S. Watanabe, “Pattern recognition: Human and Mechanical”, Wiley & Sons, New York, 1985.

[61] R.O. Duda, P.E. Hart and D.G. Stork, “Pattern classification”, Wiley & Sons, New York, 2001.