Learning Driving Patterns to Support Navigation

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of

Doctor of Philosophy in Computer Science

in the

University of Canterbury

by

Dejan Mitrović

University of Canterbury

2004

Acknowledgement

I wish to thank the Road Safety Trust for the scholarship which enabled me to start with the research presented here. My supervisor, Prof. Wolfgang Kreutzer and staff at the University of Canterbury Computer Science Department provided invaluable help in various stages of my work.

This thesis would not have been possible without support I received from my family. Tanja, Anja and Miša gave me courage to start, and strength to finish this work.

This thesis is dedicated to my late father.

Dejan Mitrović

January 2004,
Christchurch, New Zealand

Abstract

Experience is a significant source of knowledge for any human activity. Knowledge about past failures may help to avoid similar failures in the future, while repeating or even improving successes. Driving is a complex and dynamic activity, and the extensive previous experience provides great help when some important decision has to be made quickly.

This thesis proposes and demonstrates methods for learning driving patterns and their use in supporting driving navigation tasks. Driving patterns are sequences of events in the traffic system repeating over time. We developed a framework of the driving warning system based on the learned driving patterns. The learning part of the proposed system builds and maintains the model of the traffic system. The system also predicts the most likely future events. The predicted event is compared with the actual event and if/when driver's behaviour becomes significantly different from "usual", appropriate warnings may be generated.

We developed a hardware system for data acquisition from navigation sensors. We used off-the-shelf components and a standard software development environment to create an inexpensive but reliable platform for our experiments.

Neural networks are computational methods developed to mimic human information processing. In this thesis we present experiments in vehicle movement prediction, using data from a small number of sensors. We tested prediction capabilities of multi-line perceptrons, recurrent and time-delay neural networks. We found that all neural network architectures have good performances for short-term vehicle movement prediction. However, the prediction error becomes significantly higher when predicting events further in the future.

Here we also present the new method for driving event recognition based on hidden Markov models. The data from a very limited set of sensors is collected and transformed to observation sequences representing driving events. For each event type we wish to recognize, one hidden Markov model is trained with observation sequences of this type. Sequences representing test events are applied to all models, and the model with the highest probability indicates the type of event for each sequence. Experiments performed show that the proposed method has a very high recognition rate.

We developed a system to predict future driving events based on recognized event types and other data collected from sensors. This system was able to successfully predict future events for previously used routes and to detect when an unexpected event has been experienced.

Table of Contents

List of Figures

List of Tables

1. Introduction

It is human nature to travel and explore new environments, but with the advent of the industrial revolution in the nineteenth century, domestic travel became necessary as the work place, schools and food markets became separated from the places where people lived. Transportation of manufactured and raw materials also increased. To provide greater mobility for people and products, animal, and later machine power, was used to provide faster transportation and enable longer distances to be travelled.

We have always expected advances in transportation to solve traffic problems. At the end of the nineteenth century major cities such as New York were deadlocked from the huge number of horse-powered carriages. Other significant problems were the enormous amounts of animal manure on the streets (four million pounds per day in New York), dust and bacteria causing respiratory diseases, horse runaways, bits, hits, etc. It appeared that motor vehicles would solve all these problems.

However, at the end of this century, a very similar picture is present on the streets. Every day, during peak hours, about half of the vehicles on U.S. expressways slow to a crawl. Vehicle emissions are a major cause of urban pollution and ozone layer depletion, and cause a number of serious illnesses. In the U.S. alone tens of thousands of people die each year from fatal transport accidents and about three million people are injured. Such statistics are not unique to the U.S.; it is estimated that Bangkok loses around 35% of its annual economic output because of congestion problems [Gibbs, 1997].

Superficially, a model of a transportation system does not appear to be complex. The three major concepts: drivers, vehicles, and environment are shown in Figure 1.1.

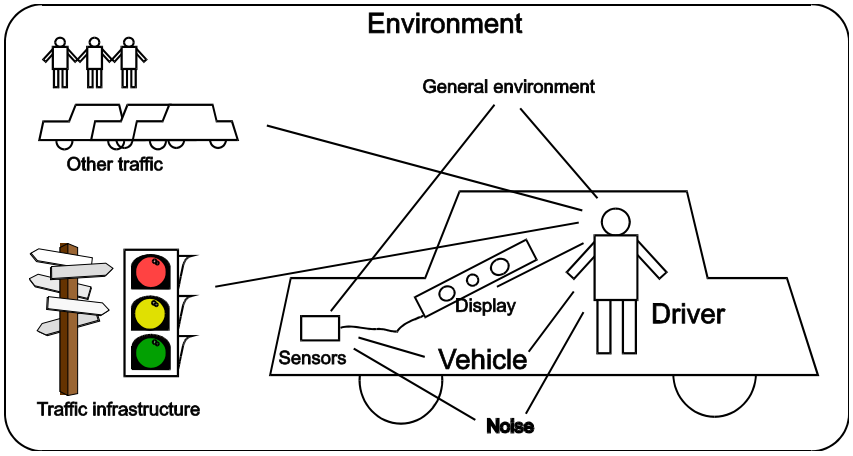

Figure 1.1 Three major concepts involved in transportation.

A driver is a human or artificial agent that controls the vehicle and navigates it through the environment to the desired place. Most of the drivers in current transportation systems are humans, because navigating and controlling vehicles requires a high level of intelligence. However, in recent years artificial drivers have emerged – most notably aeroplane auto-pilots. A large variety of vehicles are used for transportation today, but cars are certainly the most popular for everyday tasks. The environment consists of all the objects that surround the driver and the vehicle, and that influence travel. This is a huge set of objects, but the most significant are the transportation infrastructure (roads, traffic signs, etc) and other vehicles.

Transportation engineering is a research area that tries to find an optimal inter-relationship between drivers, vehicles and the environment in order to provide transport for people and products using minimal resources (time, money, injuries, pollution, etc.) However, despite the apparent simplicity of transportation problems compared to some other domains, no optimal solutions have been discovered yet.

1.1. Driving and navigation concepts

Vehicle driving is the act of operating and directing the course of a vehicle. For many humans, driving is a very easy process, perceived as a natural extension of their capabilities. However, driving is a complex decision-making process due to the multi-dimensionality of the problem (space-time), the complex relationships between the major entities involved in driving (driver, vehicle and environment) and the dynamic nature of these entities (for example, a person's driving capabilities vary depending on their age, time of day, mental, physical and emotional state, and so on).

Driving can be very complex and dangerous: increased speed requires faster driver reactions and increases the severity of injuries in accidents. Motor vehicles also have a significant drawback compared to animal-powered vehicles: animals provided additional support for the driver. Their instinct for self-preservation could prevent potentially dangerous situations from arising and keep vehicles on the road. By losing this animal support, the attention needed to control the vehicle increased enormously. Initially, the changes introduced by motor vehicles were so great that many car owners employed professional drivers and some car makers predicted that the car market worldwide would not be above one million, as not many people could be trained to drive cars.

Navigation is the science and technology of finding the position, course, and distance travelled by a ship, plane or other type of vehicles. The problems with navigation can be summarized as: Where am I? Where am I going? How should I get there? These questions appear to be simple, but in some situations it is hard to give the correct response.

Most of the time while driving, drivers solve navigation tasks. A navigation task is a decision-making procedure, where at least one of the input or output parameters have spatial properties, and are related to the current or future position or attitude of the vehicle. Navigation tasks can be classified into planning, guidance and reactive. A planning task uses a global model of the environment to plan future navigation steps in order to fulfil a previously set goal. Guidance navigation tasks (also known as tactical) execute high level commands (drive along a street, turn right, etc.) and send more accurate positional information to the higher-level tasks. They involve selecting and executing manoeuvres to achieve the short-term objectives. Reactive navigation tasks

(such as stopping in the front of an unexpected obstacle) are simply reactions to perceptions without any planning or need for a model of the environment.

The higher-level navigation tasks require the availability of a more complete environmental model, but reactive navigation tasks only require very local environmental data to be available, as displayed in Figure 1.2.

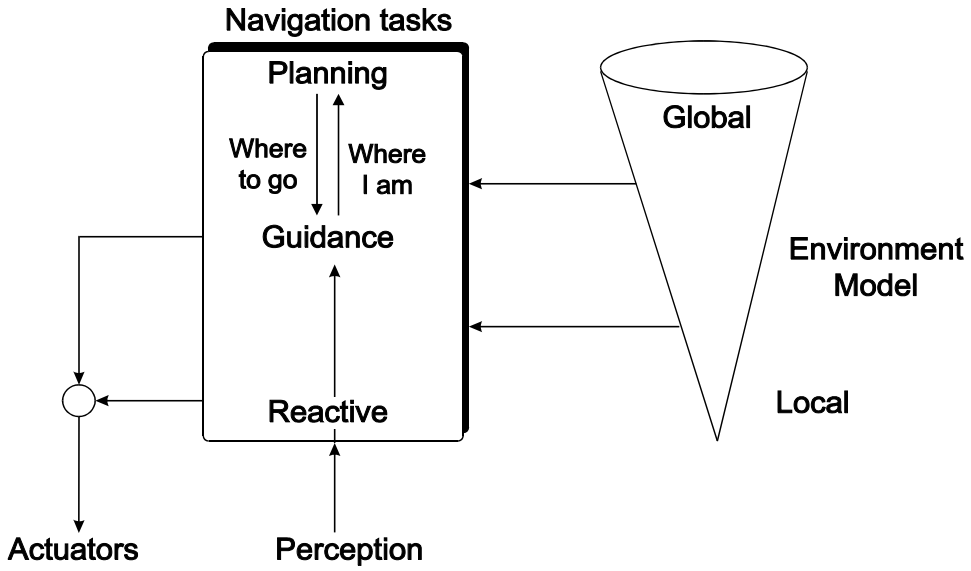

Figure 1.2 Navigation tasks and environment model.

There are many different navigation tasks involved in driving. Some are executed independently, such as parking, for example. However, the majority of driving navigation tasks are executed in parallel. Typically, a human driver simultaneously thinks about the fastest and shortest way to the destination, about the immediate traffic conditions, and maintains an appropriate distance from the vehicle in front. These navigation tasks have a strong mutual influence. If someone finds the number of cars in the neighbourhood to be very high, the selected route may be changed to include higher capacity roads. Examples of navigation tasks executed during driving are displayed in Figure 1.3.

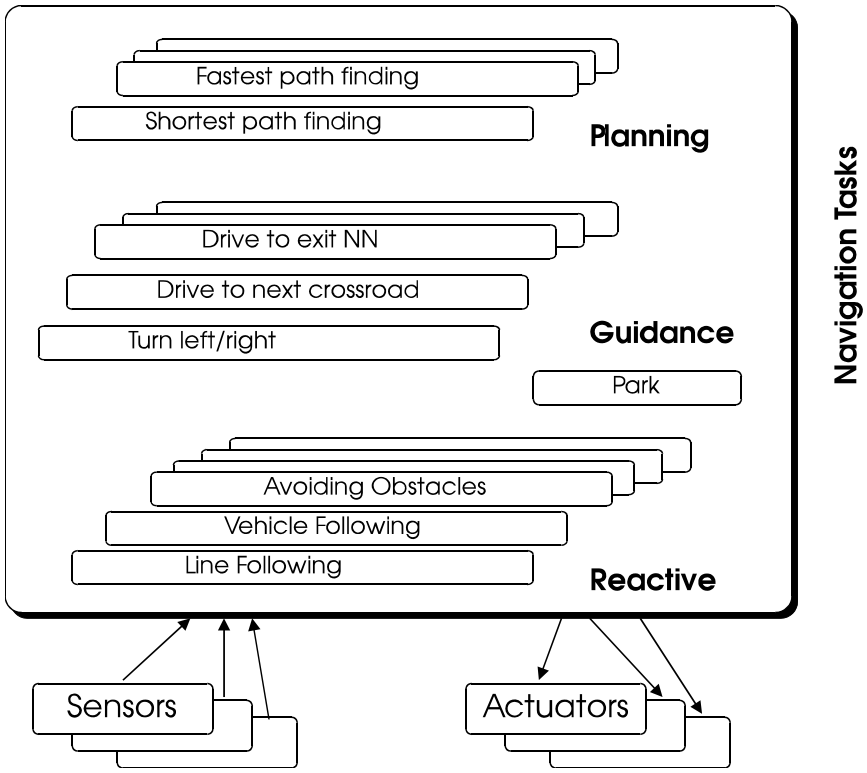

Figure 1.3. Examples of navigation tasks executed during driving.

1.2. Research in this thesis

Driving errors are the major cause of driving accidents. Some driving errors are caused by the use of drugs or alcohol, or lack of driving education, for example. Many countries have large educational programs for dealing with such causes. However, in some cases the driver makes a mistake which does not have easily recognizable external causes, and which he/she usually does not make. Using the sporting terminology, we will refer to such errors as *unforced*. These errors, like driving too fast for road conditions or failing to give way or stop, are responsible for about 25% of total driving accidents in New Zealand in recent years which were fatal or caused injury [LTSA, 2004]. For comparison, accidents related to use of alcohol caused 18% of deaths and 8% of injuries [LTSA, 2004].

We believe that unforced errors could be prevented with a help of computer based intelligent systems. The main postulate of our research is that the use of learned driving patterns can help a driver to avoid potential accident-causing situations by detecting circumstances which he/she have not encountered previously. This could be achieved by comparing actual events with events predicted from previous successful experiences. A timely warning about detected unexpected situations may help drivers to avoid unforced driving errors and improve road safety.

The main objectives of the research presented in this thesis are to demonstrate that it is possible to learn driving patterns from previous driving experiences; to demonstrate that the successful prediction of future driving events is possible based on learned driving patterns, and that it is possible to detect potential problems; to develop a model of a system which would help a driver to avoid potential errors.

Many research projects in intelligent transportation systems use very complex systems, require huge amounts of money and, as such, are out of reach of the majority of research institutions. An additional, but also important, objective of our research is to prove that equally interesting results can be achieved even with limited resources. Driving patterns can be inferred (learned) easily using standard machine learning techniques and inexpensive and widely available navigation sensors can be used. Although the model of driver behaviour obtained from a small set of sensors may be crude, we believe that it contains most important aspects of driving.

We are aware that the developing and implementation of the complete system for driver support is far more complex then we may address with resources available. For this reason our research does not address issues which we find marginal to the main topic of this thesis. For example, we will not elaborate on the use of proposed methods for autonomous driving, even we though believe that most techniques used in driving support systems are likely to be required for autonomous driving systems. We also believe that successful navigation support requires integration of many different techniques. However, this will not be further elaborated in this thesis.

This research was enabled by the Road Safety Trust of New Zealand. They provided a scholarship, an annual fund for basic hardware and software, and funding for important conferences. Conference attendance was also supported by the Computer

Science Department of the University of Canterbury, New Zealand, and Trimble Navigation New Zealand.

1.3. Contributions of this thesis

The most important contribution of this thesis is the proof that it is possible to detect circumstances which are not encountered previously. We will provide proofs for the following:

- Successful inference of driving patterns using simple and standard navigational sensors and machine learning techniques is possible.
- Future driving events can be predicted from previously encountered ones.
- Previously unseen events can be detected by comparing predicted and actual events.

More specific contributions of this thesis are:

- A novel, reliable and robust method for driving event recognition by hidden Markov models was developed. Hidden Markov models are a well-known machine learning technique, and have been used by other researchers in vehicle navigation, although with different goals, and much more sophisticated sensors and systems were used.
- A method for long-term prediction of driving events recognized by hidden Markov models is developed. It compares the currently detected sequence of events with previously seen sequences and generates hypotheses. The hypotheses are evaluated and the one with the largest similarity is chosen to predict the next event. Our experiments proved that this method is capable of measuring the similarity of event sequences, and detecting events which are not previously seen in this context.
- We proposed a framework for a system for driver's assistance based on situation model built from recognized events. Most existing intelligent transportation systems only mimic the behaviour of a human driver, which makes them vulnerable in situations which are not common. The proposed

framework is novel in that it explores previous experiences in order to detect unseen events.

- A data acquisition system based on inexpensive navigation sensors and off-the-shelf hardware components was designed. Appropriate data acquisition software for the system was developed, tested and successfully used for data collection for the purpose of this thesis and for the post-graduate research of Kerryn Offord from the University of Canterbury's Psychology Department.
- The analysis of neural network capabilities for predicting vehicle motion was conducted over a number of experiments. We confirmed that neural networks could successfully predict vehicle's movement only in the near future.
- Software for hidden Markov models' training and evaluation was developed. This software is also used for experiments in detecting patterns in video sequences at the Digital Media Warehousing Project, Computer Science Department, University of Twente, Netherlands [Petkovic, 2003].

1.4. Organization of this thesis

In Chapter 2, the motivation for this research is described in more detail. Relevant research areas are reviewed. The second chapter also provides the broad context of our research. In the third chapter the theoretical foundations of our research are presented. Related research in driver models and inferring driving intentions (predicting a driver's and vehicle's behaviour) is relatively novel, but covers a wide spectrum of technologies and applications. We will review only the most relevant results. A mathematical model for driving pattern recognition will be derived.

The data acquisition system we developed and used will be explained in the fourth chapter. The design goals are outlined and more detail about the hardware architecture is provided. Special attention will be given to the navigation sensors. The design and implementation of the data acquisition software will also be elaborated upon.

Our experiments with neural networks are described in Chapter 5. We present basic neural network concepts. The neural network architectures recommended for time-series prediction are introduced. We also review relevant research in neural network

prediction in transportation applications. The results of short-term vehicle motion prediction by neural networks are quantified at the end of this section.

At the beginning of the Chapter 6 a short tutorial on hidden Markov models is provided. Our method for driving event recognition by hidden Markov models is presented and our experiments explained in detail. We also present our method for long-term prediction of driving events, including the practical implementation and results of the experiments. A summary of the results and conclusions are presented in Chapter 7. Suggestions for further research are included.

2. Motivation

The Twentieth Century has witnessed a number of important scientific and technical advances. Each has improved human life, and made a large impact on society. Our parents were born in a time when electricity was still a kind of miracle, and now our generation cannot imagine everyday life without it; our children are even scared in the rare circumstances we are without power. For the first time in human history the rate of technological change is clearly visible not inside one's lifetime, not even in the time span of a decade, but within a single year. Computing devices bought one Christmas are generally regarded as outdated the next. Most newspapers, magazines and Internet web sites have dedicated columns for scientific and technological news, as their popularity and importance guarantee a large and steady readership.

Numerous technical and scientific advances provided the motivation for the research presented in this thesis. Those that had the largest influence on this research are discussed. This chapter also presents the technical context for understanding this research and for measuring the significance of its contributions.

It is difficult to choose the most important technical or scientific advance of the Twentieth Century: the invention of the aeroplane, radio or TV, the building of the first computer, the discovery of human DNA or our flight to Moon. Each of these, and many other inventions, are very important steps in the development of human civilization.

There are five technical areas that are important for this research and that provided the motivation for it. The first is the amazing advances in computing devices, which paved the way for many other developments. As presented in Figure 2.1. increased computing power enabled the development of new devices and sensors, and also new software tools, such as artificial intelligence and machine learning. Together, these influenced a number of application domains, among which, car navigation systems and autonomous robotics are the most relevant. In the next section more detail about the state-of-the-art in these domains will be presented.

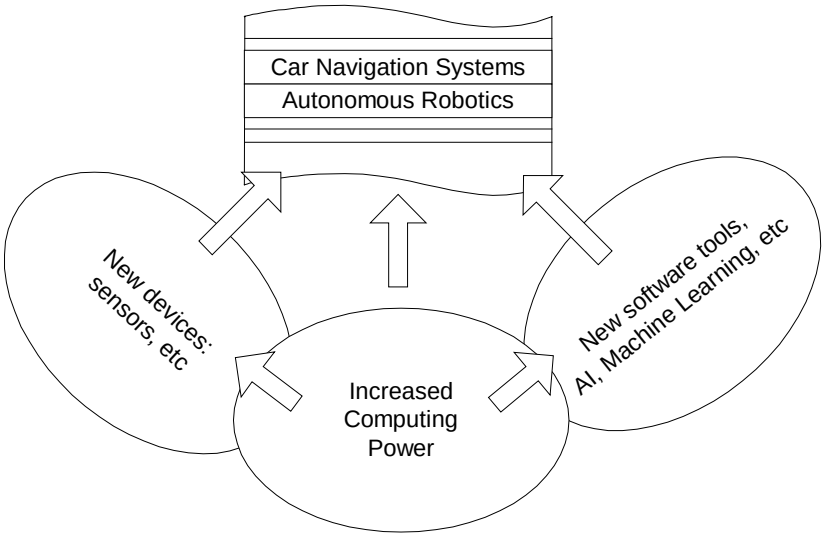

Figure 2.1. Technical areas which provided motivation for research presented in this thesis.

2.1. Computers for things that think

During most of human history, very simple mathematical tools have been sufficient for everyday life. However, scientific and technological advances in the early stages of the industrial revolution required much more complex mathematical tasks to be solved. It became obvious that humans, regardless of their proficiency in mathematics, were not fast or accurate enough for many of the calculations required. This paved the way for the first mechanically operated calculating machines. Blaise Pascal developed the first hand powered adding machine in the 1840's. A few decades later another great scientist, Gottfried Leibnitz, constructed a mechanical multiplier.

The increased calculation power of mechanical devices was a large help to solving some tasks. However, in order to increase the applicability of these devices, they needed to be programmable. In 1801, Joseph-Marie Jacquard built a loom that wove patterns stored as punched holes in a series of hardwood sheets. This was the first "programmable" machine in the world. Charles Babbage created the first programmable calculating machine in 1842. In the following decades faster and more powerful electrical devices gradually replaced mechanical devices.

Most commonly, a computer is defined as a programmable electronic device that can store, retrieve and process data. At the beginning of World War II a number of devices which broadly fulfilled this definition were developed by Konrad Zuse in Germany, John V. Attanasoff at the University of Iowa, the British team for the breaking of the Enigma code led by Alan Turing, and the ENIAC project at Penn State University. These people can be regarded as the parents of the computers we know today.

Computers have had a fast and steady development. In 1953 IBM produced Model 701 as the first commercially successful computer. In 1958 Control Data's 1604 was the first transistor powered computer, and with the introduction of integrated circuits, the first minicomputer (DEC's PDP-8) was introduced in the mid-sixties.

Until this stage, computers shared the fate of many innovations that improved some aspects of human life, but had little impact on the everyday activities of ordinary people. However, computers continued their fast pace of development following

Moore's law – computation speed and size of the memory double every eighteen months. Observed in the mid 1960's, Moore's law has held remarkably true for more than a hundred years since Hollerith's tabulating card machine was used in the US census in 1890.

The introduction of microprocessors thirty years ago, and the first personal computers, provided cheap and affordable computing power for a wider audience. Personal computers soon became consumer articles, and mass production quickly and significantly dropped their price, further increasing the number of potential users.

Further advances in microelectronics made it possible to combine the processor, various types of memories and input/output controllers into a single chip, called a microcontroller. Microcontrollers further reduced the price of computing power and made it possible to embed them into various devices. This greatly improved many existing devices (such as washing machines) and enabled many new types of devices to appear. With advances in personal computers and microcontrollers, the scene was set for the most significant change in human civilization. As animal and machine power greatly increased human capabilities to build, travel and produce, computers increased human capabilities to calculate, make decisions and think. Due to the exponential increase in computer performance, some people predict that it will reach the information processing power of the human brain in the next twenty years, and may quickly become the dominant intellectual force on Earth.

However, even at the current level, computers are not exploited to their full capabilities. In the year 2000 alone, 385 million microprocessors and 6.4 billion microcontrollers were produced, almost two for every person on Earth. Tens and hundreds of computing devices surrounds an average family in the developed world. The total computing power of these devices is larger than required for the functions that they execute. Most computing devices are not used for the majority of time. Take for example modern microwave appliances: their microcontrollers are used for only a few minutes per day on average. In 1969, Apollo 11 computers used 150KB of on-board memory to take people to the moon and back. Today, at least 500KB of memory is used just to keep the CD player from skipping tracks.

The wide availability of cheap (or free) computing power around us has led to a number of research projects to develop architectures and technologies that will employ

various computer devices to provide information where and when it is required without it being managed by the user. One of the most interesting projects of this kind is MIT's "Things That Think" (www.ttt.mit.edu). The basic motivation for this project is that we must expect more from our environment and that thinking is not the result of a small set of profound rules, but comes from many things that each know something. The idea is to build many smart devices in which many agents will collectively interact with many different kinds of information and create latent intelligence that may help the user.

2.2. Affordable navigation sensors

Until the twentieth Century, the term "navigation" referred mainly to guiding ships across the seas. Even the origin of the word "navigation" in the Latin means "ship guidance". However, today navigation has a much wider usage and also encompasses travel on land, air and in space. Animals and humans are very good at land navigation. They easily use natural features such as rivers, streets and hills to find the appropriate route to the desired place. This technique is known as landmark navigation.

The first major obstacle in navigation for humans was when they needed to travel across oceans with no visible features that could be used as landmarks. Only celestial objects were visible, but their position was not constant. Consequently, navigational instruments and complex mathematical tools had to be developed to calculate the ship's position and attitude. A quadrant (similar to the sextant) was invented in 1731 to measure the positions of celestial bodies. In the next thirty years the accuracy of chronometers improved, with Captain James Cook's voyages in the 1770's being the first to prove the merits of scientific navigation.

Advances in navigational instruments have been steady but slow. Due to the mechanical nature of the instruments, very fine manufacturing processes were required to obtain the required accuracy. For the same reason, navigational instruments were very expensive and affordable only for large ships and aeroplanes. As in many other domains, advances in electronics and computers greatly influenced the development of navigation sensors in the last few decades. New navigation sensors use radio, magnetic, optical and other techniques to provide the necessary data. Even where the mechanical nature of the sensor had to be preserved, its size today is many orders of magnitude smaller, as it is

built using micro- and nano-technologies. Finally, a set of navigational sensors that can provide the position of any vehicle is approximately a thousand times cheaper than a device with a similar accuracy fifty years ago.

Navigation sensors are classified into absolute and relative. Absolute sensors are capable of providing the sensor's position with respect to a predefined reference system, usually a geodetic coordinate system. They are implemented as land-based or satellite radio–navigation systems, depending on the position of radio beacons (Earth's surface or Space). Land-based navigational systems were developed during and after World War II for ship navigation and some are still in use. LORAN–C [Frank, 1983] is a pulsed hyperbolic system operating in the 90–110 kHz frequency band. It is based on the measurement of the time difference between pulses from two transmitters. It is known that all positions with a particular pulse difference lie on a hyperbola. If we measure the time difference for pulses, we know that the vehicle is on a particular hyperbola. The process can be repeated with other pairs of stations and the position of a ship is calculated to be at the intersection of the found hyperbolas. The repeatable accuracy of LORAN–C system is 18 to 90 metres. Many countries provide this, or a compatible, service so it covers large parts of the world. The Omega system [Sage, 1983] is similar in nature, but uses very low frequencies and provides world wide coverage with a lower accuracy (two to four nautical miles).

Very soon after the launch of the first artificial satellites, satellite based navigational systems were developed. The first deployed satellite system was Transit [Sage, 1983]. It consisted of six satellites in polar orbits at an altitude of 1075km. Receivers measured the Doppler frequency shift for the signal transmitted from the satellites, whose positions were known, and this was used to calculate the position of the vehicle with an accuracy of 35 to 100 metres. The Transit system had some problems, particularly large gaps in coverage, but was very successful and worked for more than thirty years.

The success of the Transit system paved the way for the most successful navigational system in history – the Global Positioning System [Parkinson, 1996]. The U.S. Department of Defense developed the Global Positioning System (GPS) primarily for military use. GPS consists of 24 satellites that broadcast predefined signals. An end-user device receives signals from three or more satellites, calculates the distance from

each satellite and finds the position of the receiver as an intersection of spheres constructed from known satellite positions. The accuracy of the GPS receivers is better than 25 metres and can be increased by using inexpensive correction tools. The most sophisticated systems have a relative precision of a few centimetres.

The GPS system requires very complex signal processing and mathematical calculations. Fortunately, it was deployed at the same time as major advances in telecommunications and computing occurred. In the span of only one decade, the technology to build receivers, which once was affordable only to the U.S. military, became accessible to common users. The simplest receivers became cheaper than common small sized TV sets, and GPS is soon to become standard in mobile phones and handheld computers.

Relative navigation sensors provide information about a vehicle's position or attitude with respect to the previous position and attitude, or to some other object. *Dead reckoning* is an old, but simple and widely used navigational method. The position is calculated on the basis of the distance travelled and headings since the last observed position. Possible errors caused by wind, currents, or sensor inaccuracy must be taken into account. The odometer, a car's mileage counter, is the simplest and the most widely used navigation sensor. The advantages of odometers are their permanent availability and self–sufficiency. However, dead reckoning is very error–prone, and positions obtained from odometer-based sensors can be used only for short periods with acceptable accuracy.

Using odometers on sea and in air is much harder, as precise measurement of the travelled path is not easy. *Inertial navigation systems* have been developed for use in ships and aeroplanes. This method uses accelerometers and gyroscopes to measure the rate of rotation and acceleration of the vehicle. The distance and direction of travel are calculated by integrating (once or twice) measurements from sensors. Inertial navigation systems usually have three accelerometers and three gyroscopes to measure positional changes in the three–dimensional space. They have similar advantages and disadvantages as dead reckoning systems. However, the error in inertial sensors does not depend on outside factors, and can be minimized by careful design and production. Due to their high cost, inertial navigation systems have been used only for expensive

vehicles. For example, a high–end system for commercial airliners with a typical error of one mile per hour of operation costs more than U.S. $50,000.

Electromechanical accelerometers are based on a body of relatively large mass attached to the sensor case by an elastic support consisting of a dumper and suspension spring. A pick-off system is constructed as a coil positioned on a body which interacts with the magnetic field of the permanent magnet attached to the case. Pick–off detects changes in the body's position with respect to the accelerometer case. Electro–mechanical gyroscopes are also based on a body with a relatively large mass. It rotates at a high angular velocity. From playing with flywheels, we know that, if the rotating body is properly suspended, the spin axis will tend to maintain its orientation with respect to a fixed line in space. However, if the rotational motion perpendicular to the body rotation is applied to the gyroscope, its spin axis will change. An appropriate angular pick–off system should be able to detect this angular deviation and produce an output signal proportional to the applied rotational motion. The advances in electronics and related fields made significant changes in the design of accelerometers and gyroscopes. The new generation of accelerometers and gyroscopes are built using micro-machining processes, making production very inexpensive and the size very small. Such new generations of inertial navigation systems are more than an order of magnitude cheaper than those available ten years ago.

Another important class of navigation sensors are proximity devices, which can detect a vehicle's distance from other vehicles and objects. Radars were the first types of proximity devices, and were based on the radiation and reception of microwaves. For a long period of time, the price of radars was affordable only for military and government users. However, in the last decade, many new devices based on infrared, ultrasonic and microwave radiation have become available and affordable for the public. Recently devices for detecting a car's distance from other cars at the front or back have become available as a solution for the problem of parking in narrow city environment.

Modern cars are typically equipped with many sensors of various types. Dedicated navigation sensors are only used in the higher class cars, but many sensors used for other purposes (such as accelerometers that fire air bags) could also fulfil navigational tasks. Modern cars also contain a number of micro-controllers, whose price has become lower than the price of dedicated electronic controllers. Therefore plenty of information

processing power is available in a modern car. It is reasonable to expect that these sensors and processing units could be better exploited to provide additional value to the driver.

2.3. Artificial intelligent beings

Great efforts have been made throughout history to understand the behaviour of intelligent beings and, most importantly, humans. Many great philosophers and psychologists have dedicated their lives to answer this fundamental question: how can a small and slow brain perceive and understand a world that is far larger and more complex? As yet no complete answer has been found. The invention of the computer, however, provided a convenient tool, not only to study human intelligence, but also to attempt to create artificial intelligent devices or programs, a field known as *artificial intelligence* (AI).

The first thoughts of creating artificial intelligence by computers began very early in the development of the computer. McCulloch and Pitts laid the foundation for neural networks in 1943 and Turing wrote about "computing machinery and intelligence" in 1950. His "Turing test" was designed to decide whether or not a machine is intelligent [Turing, 1950]. In this test, the human judge must decide if the dialogue is with a machine or another human. Communication is conducted through written messages over terminal lines. Turing predicted that computers would be able to pass his test in the early 21st Century.

Early development in AI focused on problems that are regarded as difficult in the scientific community: chess playing, solving mathematical theorems, and reasoning in medicine and chemistry. In the late 1950's the team from Carnegie–Mellon University created the General Problem Solver [Simon, 1958]. This program used recursive search techniques to find proofs for various mathematical theorems. It managed to find proofs for some previously unsolved theorems. This led authors to the conclusion that a computer would be the world chess champion by 1968. Early success was also reported in a number of other AI domains: the programs "Analogy" [Evans, 1968] and "Student" [Bobrow, 1968], developed at the MIT, were able to outperform most high school students in solving geometric analogy IQ tests and algebraic problems. Stanford's

DENDRAL program [Buchanan, 1978] was the first expert system, able to answer questions about chemical compounds. Greenblat's chess program [Greenblat, 1967] defeated Herbert Dreyfus, an AI critic who strongly doubts the ability of computers to play chess.

Early AI programs solved a number of complex problems. However, a great many, including some apparently trivial problems easily solved by a pre-school child, were beyond their capabilities. This caused considerable disappointment in the AI research community, and many government agencies stopped funding such research projects during the seventies.

In the eighties, more powerful personal computers and workstations significantly changed the field of AI and a much wider group could now afford to participate in AI research. In smaller centres, the research focus was on smaller and easily attainable goals. The first commercial expert system called R1 [McDermott, 1982] began operation at the Digital Equipment Corporation in 1982. In 1986 the back-propagation algorithm was re-invented and new life was given to the field of neural network. Speech and optical character recognition programs became commercially available and inexpensive. Real world applications replaced the "toy experiments" of the fifties. As many new companies started producing programs that contained intelligent behaviour, the commercialisation of AI become more widespread. Today, basic AI techniques have become so well known and robust that manufacturers readily include them in their products to provide added value for customers. For example, "intelligent" washing machines are able to precisely set up a washing program according to the user's preferences and the state of the clothing. The user is not required to consult manuals or to limit a machine with a range of capabilities to only one or two options. Another example are mobile phones with voice dialling – just a few years ago speech recognition was state-of-the-art in AI research; today it is part of an inexpensive consumer product. Another important breakthrough in AI came when the Deep Blue computer [Hsu, 2002] defeated the world chess champion Gary Kasparov in an exhibition match in 1997.

A crucial ability of any intelligent being is the capability to learn. In AI research, the methods and techniques for learning are called *Machine Learning*. Learning systems are much more capable, as they can adapt to data that are out of the scope for which the system was designed and may work properly even in unforeseen environments. One of

the earliest learning techniques was the neural network. A perceptron style network and Adaline [Widrow, 1960] were the first to provide learning capabilities through trainable threshold units. Symbolic learning techniques were based on decision trees, such us ID3 and its derivates by Quinlan [Quinlan 1979], and the candidate elimination algorithm by Tom Mitchell [Mitchell 1977]. Statistical learning methods, such as hidden Markov models [Baum, 1966], enabled successful learning from domains that have large quantities of temporal data, while reinforcement learning techniques have been successfully used in complex domains such as game playing and robotic control [Sutton 1998]. Genetic algorithms use an evolution metaphor to learn [Friedberg, 1958], by applying cross-over and mutation on "genes" (basic strings or system parameters).

Advances in artificial intelligence and machine learning in the last few years have provided tools for the creation of intelligent and virtual environments around humans (in a home, car, shop, etc). There humans may interact with other humans and with virtual, interactive characters in a way predicted by Turing. Together with advances in telecommunications, this may overcome many limitations of human civilization and pave the way for a much more advanced and prosperous society.

2.4. Car navigation systems

Obviously, humans have needed navigational guidance since the invention of the vehicle. Various navigational instruments and techniques (odometers, magnetic compass, etc) have long been in use. However, the introduction of such instruments into the modern automobile was very gradual. According to Chinese legends, vehicle navigation systems were invented in China around 2300BC. The first was a south-pointing carriage, that used a gear-based differential odometer to keep the direction of the figure's hand pointing towards the south. Another was a "li-recording" drum carriage. It had a set of gears that moved two wooden figures to strike the drum after each one or ten lis traveled (one li is approximately 500 m).

The first attempts were made before World War I, with mechanical route guidance devices. These devices, driven by an odometer shaft, provided real-time route instruction to the driver. Route information was recorded by punching a moving tape or rotated disk or drum. During World War II, The U.S. army developed an

electromechanical vehicle trajectory plotter based on a magnetic compass and odometer. The invention of computers paved the way for dynamic navigation systems, such as project ERGS in the Eighties [Rosen, 1970]. It consisted of a network of short-range beacons at road intersections. These provided a communication link between the vehicle's device and a central computer. The central computer calculated the best route and guidance instructions for the car. These instructions were transferred back to the car's device, which presented them to the driver.

As computers become widely available in the Eighties, a large number of research projects related to vehicle navigation began. Computers enabled researchers to cast their nets more widely. Research topics covered driving support systems, automated highways and many other aspects of transportation. The complexity of the domain and the availability of computational power led to a more widespread usage of artificial intelligence techniques in such systems. The term *intelligent transportation systems* denotes such systems.

An important event for car navigation systems was the introduction of microprocessors, which enabled cheap computers to become car components. Microprocessors enabled the development of autonomous navigation systems such as Etak Navigator in 1985 [Teleatlas, 1989] and Philips CARIN in 1987 [Knowling, 1999]. These systems were based on dead reckoning. As previously stated, dead reckoning sensors have a large system error that grows over time. The Etak Navigator and CARIN systems solved this problem by using digital map matching. They compared the position calculated by the dead reckoning system with data from a digital map and re-calibrated the dead reckoning system at each turn or other significant driving event.

With the development of the global positioning system, cheap GPS receivers are readily deployed in car navigation hardware and software. Today it is very easy to find a vehicle's location with an accuracy appropriate for car navigation. The problems with limited GPS coverage in the "urban canyons" were successfully solved by the application of inexpensive inertial sensors and intelligent map-matching algorithms. A number of car manufacturers (particularly in Japan, the U.S.A. and Europe) provided car-navigation systems as a standard component. For example, General Motor's OnStar system [OnStar, 2003] is available in 32 of a total 54 models produced by GM. Inexpensive car navigation software is available for laptops and personal digital

assistants. Complete systems that include a GPS receiver, a road database, car navigation software and a device with a small LCD display can be purchased for a few hundred U.S. dollars.

Another very active research area in the application of computers in driving is known as *driver support systems*. Driver support systems (also known as driver's warning assistants) are automatic co-pilot systems that monitor the driver, the vehicle and its environment in order to provide timely information about incoming major decisions and potentially risky situations. Driver support systems use artificial intelligence techniques to cope with the complexity of the domain. Driver support systems that are oriented to navigation either support reactive navigation tasks (collision avoidance [An., Harris, 1996]) or are computer vision based which makes them very complex [Rombaut, 1993]. Such systems are not yet commercially available, although, some achievements from this research have been incorporated into modern car navigation systems.

In parallel with the development of the car navigation systems, many other applications of electronics in cars have been developed. As can be seen from Figure 2.2,

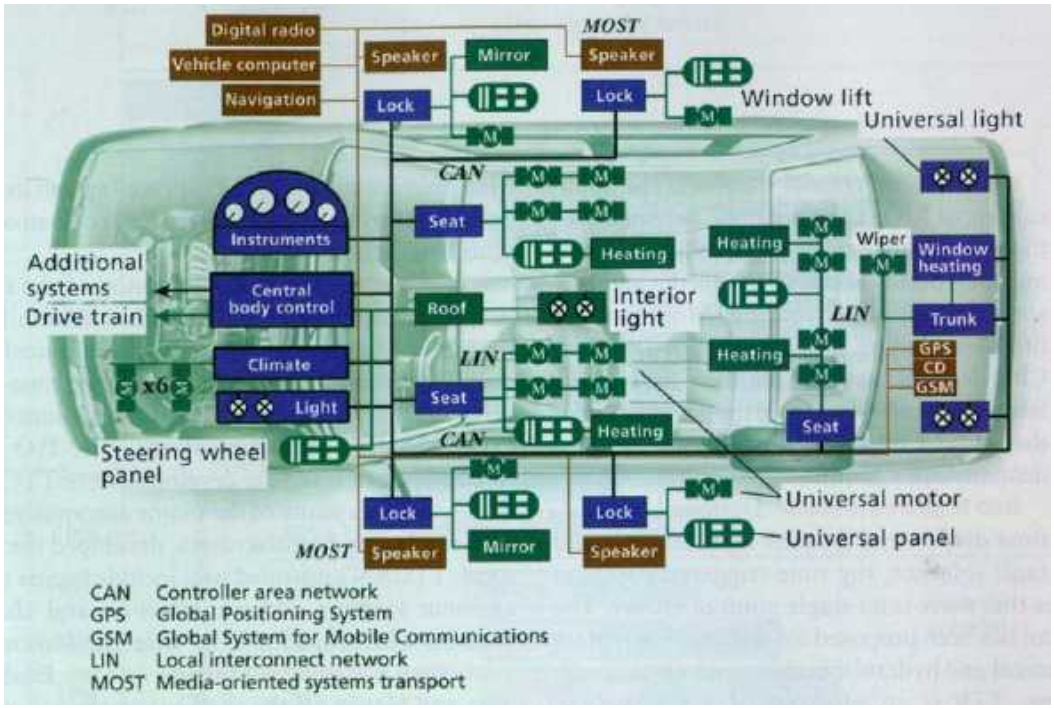

Figure 2.2. The electronic components in a modern car [Leen, 2002]

the modern car has many sensors, actuators and microprocessor based control components. To reduce the need for wiring (which otherwise could reach up to 50kg) automotive control networks were developed. The most widely used is CAN (controller area network) developed by Bosch in the mid-1980s [Nath, 1998].

Recent advances in wireless communication and the Internet have opened new possibilities for car navigation systems and other electronic components for vehicles. The fusion of these technologies is called *telematics*. Car systems are today able to access various Internet services and to obtain information on the driver's behalf. They can provide the user with the best possible route, not only with respect to the existing road network, but also to current road conditions, traffic situations, etc. Drivers can be informed whether the vehicle has moved without their knowledge, and emergency teams can be alerted as soon as an air bag is fired in the car. The entertainment aspect of telemetric products is also very important, and often components for audio, video, gaming and Web browsing are available as part of the car's telematics systems. The Frost & Sullivan marketing company predicts that the telematics market is expected to achieve revenues of U.S.$7.2 billion in 2005 (ITS World Jan/Feb 2000, p. 26).

Advances in car navigation systems and lower prices have led to the development of *personal navigation systems* that can provide the location of a person. It has become a legal requirement in the U.S.A. that, over the next few years, all mobile phones will provide positional information for search and rescue purposes. At the same time the current market trend in mobile communication is to provide continuous Internet connection. A combination of these technologies opens great possibilities for various location-based services. The user of a mobile phone will soon be able to retrieve the positions of the nearest ATM machine, supermarkets, etc. In such an environment, car and personal navigation becomes only a small segment of all possible applications. With a huge potential market (billions of people worldwide having mobile phones), the further development of car navigation systems will be faster than ever.

2.5. Autonomous robots

Since the beginning of history, humans have tried to use elements of the surrounding environment to make life easier or to perform tasks more efficiently. In many cases,

human features have been used as templates for building machines. Today's automatic devices that perform functions normally ascribed to humans, or devices that have human-like form are known as robots. The word "robot" first appeared in Karel Capek's play "Rossum's Universal Robots" [Capek, 1921], about dehumanization in a technological civilization with artificial workers. The Science Fiction author Isaac Asimov also wrote about robots. He proposed the basic "Laws of Robotics" setting the ethical basis for a society that consisted of both humans and robots, many years before the first robots were even built [Asimov, 1950].

Robots as they are known today, do not really imitate humans or other living forms except in the limited aspect of digital dexterity. One root of their development lie in an effort to automate some or all of the operations required on the factory floor. This effort began in the 18th Century in the textile industry, when looms were designed to perform under the control of punched paper tapes. With the burgeoning of the Industrial Revolution, more factories sought to bring a greater degree of automation to the repeated processes of the assembly line. True robots have not been possible, however, until the invention of the computer in the 1940s and the progressive miniaturization of computer parts. One of the first true robots was an experimental model called SHAKEY, designed by researchers at the Stanford Research Institute in the late 1960s [Nilsson, 1984]. It was capable of arranging blocks into stacks through the use of a television camera as a visual sensor, processing this information on a small computer. Thereafter engineers tried to adapt robotlike devices to useful tasks. In the mid-1970s, General Motors financed a development program in which a Massachusetts Institute of Technology researcher, Victor Scheinman, expanded upon a motor-driven arm he had invented to produce a so-called *programmable universal manipulator for assembly*, or PUMA [Scheinman, 1987]. The PUMAs that resulted marked the beginning of the age of robots.

Autonomous mobile robots are capable of moving through an environment without external help, while executing some useful function. Navigation is a very complex task and requires large computing resources – only recent advances in computing power and navigation sensors have enabled autonomous mobile robots to be developed.

Early mobile robots used the *Sense-Plan-Action* control approach. Data from sensors was compared against a world model to find the robot's current position. This was used,

together with the world model and the given goal, to form a plan of the robot's future action. An action is executed through the robot's actuators. If required, the world model is updated from data received from the sensors. This approach was simple to implement, but has had a few major problems:

- The Sense-Plan-Action cycle was too slow and only simple situations and actions were possible, or the robot's motion had to be very slow.
- The world model can be very complex – which made comparisons and updates difficult.
- Too much data had to be transferred from the sensors to the planner, and, further, to the actuators.

Investigation into insects' navigation showed that insects successfully navigate without forming complex world models or high level plans. Based on these assumptions Rodney Brooks developed a *subsumption* robot architecture in 1986 [Brooks, 1986]. The subsumption architecture consists of a number of different behaviours, which respond to a particular sensor input. Behaviours are implemented as separate computing processes that run in parallel. The arbitration over which behaviour output will be executed is performed at the actuators. The first systems based on the subsumption architecture were very fast and appeared to be a major breakthrough. For example, the HERBERT robot [Connell, 1990] was developed to collect soft-drink cans. It had impressive capabilities even by today's standards, but it is not known whether it ever completed a task without a problem. Other behaviour-based robots had similar problems. A weakness in the subsumption architecture has been detected, according to Hartley [Hartley, 1991], in an interference that was required between behaviours. This disabled the modular design, resulting in systems that were too complex.

A number of researchers in the early Nineties came up with very similar control strategies, commonly known as the *three layer architecture* [Connell, 1991][Bonasso, 1991]. The three layer architecture is not derived from any control theory. However, it was successfully implemented in a number of robots and became a de-facto standard. It consisted of three control layers:

- The *Controller* (skill layer) applies sensor inputs to behaviours (skills), which in real-time generate control signals for actuators. A number of behaviours, from a predefined library, can be executed in parallel.
- The *Sequencer* (sequencing layer) selects behaviours to be applied in a concrete situation and sets parameters for behaviours. The linear sequence of behaviours provides a mechanism to execute useful high-level action.
- The *Deliberator* (planning layer) conducts high level planning without time constraints. The deliberator sends the created plan to the sequencer for execution, and/or is ready to answer the sequencer's queries about further actions.

The major advantage of the three layer architecture is that the same robot can be used to execute different tasks – the task specific program represents only a very minor part of the complete code. One good example is the robot Alfred, which successfully competed in two events at the 1993 AAAI mobile robot contest [Nourbakhsh, 1993]. One important feature of Alfred is that its controller runs on a robot control board, while the deliberator and sequencer run on a Macintosh Powerbook computer connected to the control board by a serial port. This is good argument for the modularity of the three layer architecture.

The best known autonomous mobile robot to date is Sojourner [NASA, 1997], the microrover from the Mars Pathfinder mission. This 68cm long, 10.5 kg, 6 wheeled robotic vehicle occupied the attention of the whole world for a few weeks in 1997. Sojourner was able to execute high-level commands received from Earth (e.g. go to position X,Y), to navigate around terrain hazards without human intervention, and, at the same time, to execute a number of scientific experiments and collect a vast amount of data. Sojourner was a great example of how useful mobile robots can be and what we can expect in the future.

In the same year that IBM's Deep Blue defeated the human world chess champion and when Sojourner wandered on Mars, the first RoboCup competition was held. The Robot World Cup Initiative (RoboCup) is an international research and education initiative to provide a standard problem, in which a wide range of technologies can be integrated and examined [RoboCup, 2003]. Its primary domain has been a soccer game and each year a number of teams from all over the world bring their robots to play soccer in a number of

categories. The objective of RoboCup is stated as follows: “by the mid-21st Century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup.” This goal may sound overly ambitious given the state of the art of technology today. Nevertheless, we believe it is important that such a long range goal be claimed and pursued. It took only fifty years from the Wright Brother's first aircraft to the Apollo mission that sent men to the Moon and safely returned them to Earth. Furthermore, it took only fifty years, from the invention of digital computers to the Deep Blue. We recognize, however, that building humanoid soccer players requires an equally long period and extensive efforts of a broad range of researchers, and the goal will not be met in any near term.”

2.6. Conclusion

In this chapter we presented a number of relevant scientific and technical areas that motivated the research presented in this thesis. The invention of computers and its fast and steady development has been the cornerstone for many other inventions in the last few decades. As mechanical devices enabled users to solve physical tasks that could not be solved by human labour, so computers have enabled people to solve intellectual tasks for which the human intellect was too limited. At the same time, this tool is so inexpensive and small that it can be easily integrated into many other devices, providing new possibilities. Computers enabled the invention of the global positioning system and many other navigation sensors. Many software products have been developed to employ computers for particular tasks. Among them, a prominent place is reserved for artificial intelligence which enables many machines to behave intelligently. Car navigation systems and autonomous robots are two applications of these new technologies that could make human life easier. In both areas many advances have been made in recent years, with everyday applications changing our lives.

We believe that, in many existing systems, an important aspect of intelligence is missing: the ability to learn. In the majority of existing systems, particularly in complex domains such as car navigation, intelligent behaviour is obtained by using knowledge from a human expert, encapsulated into the system. Without the ability to learn, such

systems are not able to adapt towards changing environments and circumstances. The research presented in this thesis aims to show that learned driving patterns can be used to help a driver with navigational tasks.

3. Learning Driving Patterns

The goal of this chapter is to present the theoretical framework of our research. We first present the problem that we would like to solve with this research, provide the analysis of the components and propose a theoretical solution. The traffic system model is presented, as well as the importance of driving patterns for traffic research. By learning driving patterns, a comprehensive situation model can be built. We believe that this model can be used to solve a number of navigation tasks in driving. We also propose a framework for a driving warning system based on the described situation model, as well as a few other potential applications of the model.

The results of the research presented in this chapter have been published in the following papers:

Mitrovic D., Learning Driving Patterns to Support Navigation Decision Making - Preliminary Results, *Road Safety Conference*, 1998.

Mitrovic D., Experiments in Vehicle Movement Prediction, In *Proceedings of the Road Safety Research, Policing and Education Conference*, pp. 519-526, 1999.

3.1. The traffic system

The traffic system consists of millions of vehicles and their drivers moving on thousands of kilometres of streets, roads and highways every day around the world. Numerous people also participate in the traffic system as pedestrians and cyclists, and in other roles. Besides roads, traffic infrastructure contains a number of other components, such as signs, traffic lights, road marks, and so on.

On a small scale, from an individual driver's point of view, the traffic system can be defined as a system that consists of the individual driver, his/her vehicle and the surrounding environment (which is made up of traffic infrastructure, other vehicles, and other traffic participants). The traffic system, consisting of a driver, a vehicle, and the environment, can be represented as a closed loop control system, as depicted in Figure 3.1. The driver has some predefined goal that provides motivation for the driving. In order to realize this goal, the driver acts on the vehicle (drives). As a result of this act, the vehicle changes its position relative to the traffic infrastructure and other vehicles and participants. This may cause some reaction from the other active elements of the traffic system (the other vehicle may stop to give way, for example). The driver uses his/her perception sensors to collect information about the new situation (position of his/her vehicle and the environment). The driver analyses the new situation and makes a decision about further actions according to the preset goal.

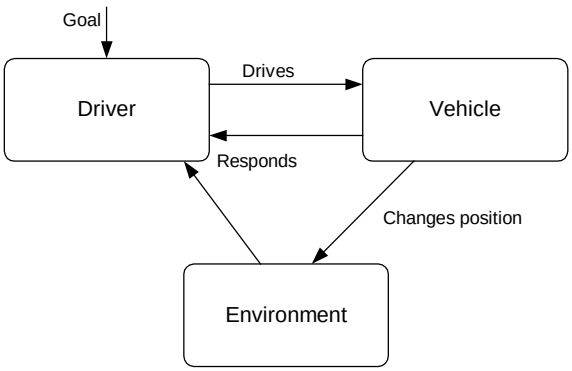

Figure 3.1. Driving control loop.

Driving tasks are executed on different levels of abstraction. The same applies to the driving control loop: on the highest level of abstraction, the driver in the control loop only deals with "coarse" elements of the traffic infrastructure: street segments and large intersections. On the lowest level, the driver deals with the immediate environment in the vicinity of his/her car.

Traffic engineers have worked hard over the last century to improve the traffic infrastructure in order to make driving easier. Roads have been widened and straightened, and many traffic signs have been added to help drivers. Vehicles have also improved, with new safety features added, including ABS brakes, new tyre designs, improved vehicle resistance to crash conditions, and so on. Particular attention has been given to the education of traffic participants. Today, traffic-related topics are a regular part of the primary school curriculum worldwide. The criteria for obtaining a driver's license have been tightened over time, and penalties have increased.

However, traffic accidents still take their toll in human lives and destroyed property. Traffic accidents are caused by the complexity of the system, its highly dynamic nature, and a number of random (stochastic) disturbances inside the system. Each day, millions of vehicles move on streets and cross paths, or meet other vehicles with a speed difference of 100km/h or more, separated by only few meters. A vehicle's lateral position can be outside its ideal placement on the road for a number of reasons: the lateral slope of the road, a random error in the steering system, a sudden wind gust, driver fatigue, or some other factor. The influence of random factors on the traffic control system is presented in Figure 3.2.

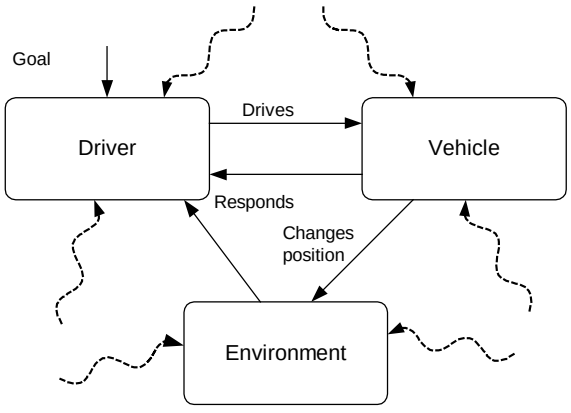

Figure 3.2. Driving control loop with random disturbances (dashed lines)

In order to eliminate or at least compensate for effects of the random processes, three general approaches are used:

- Safety margins are increased to enable the system to survive larger random disturbances. For example, wider roads allow vehicles to stay in their lane despite strong lateral wind gusts.
- Warnings are provided, wherever appropriate, so that vehicles or drivers are in a better position to react quickly to a disturbance. For example, raised median marks on New Zealand roads quickly inform a driver that he/she crossed the dividing line.
- Artificial controllers are provided to replace the human driver in executing some driving tasks. Artificial controllers should better react on random disturbance and thus improve the stability of the whole system. For example, adaptive cruise control systems can maintain the safe distance from the vehicle in the front.

Humans are the most common source of random disturbances in the traffic system. The traffic infrastructure and a vehicle's mechanical systems are well defined and built under strict standards to avoid randomness. Traffic rules are refined over years to eliminate ambiguities. However, the human decision-making process is very complex and, as such, very sensitive to external influences. Further, this process is extremely non-linear – a very small change in external parameters can lead to a large change in a particular decision. For example, the fact that approaching cars have their lights on (even in daytime conditions) could cause someone to make a different decision in his driving.

Each component of the traffic system can be changed to improve the system. Changes in the traffic infrastructure or driving rules have an impact on many drivers at the same time, and therefore represent a convenient way to influence driving control in the traffic system. However, changes in traffic infrastructure are limited by the physical nature of the infrastructure, thus making change expensive. On the other hand, drivers have learned to use existing infrastructure and rules, and each change, regardless of its long term benefits, may cause increased problems in the beginning, while drivers accommodate to the change.

Vehicle characteristics are constantly improved, making them a more reliable part of the control loop. For example, ABS breaks prevent tyre blocking on wet surfaces and reduce the probability of unexpected events, making the whole system more deterministic. However, due to the mechanical nature of vehicles, possible changes are of limited scope. The large numbers of existing drivers who must be educated to use the new technology and the high price of introducing new technologies in the manufacturing process are also major constraints.

Therefore, the possibilities for increasing safety margins are relatively limited. As a consequence, a large number of traffic research projects are dedicated to systems that may replace or augment the human driver. Some of these systems will be described in more detail in the related work section. The research presented in this thesis has a similar goal. We are developing a new technique to support the driver in solving navigation tasks.

3.2. Driving patterns

Driving patterns are sequences of events in the traffic system repeating over time. Driving patterns are caused by:

- Regularities in the traffic infrastructure. For example, most urban areas are developed as regular grids of streets, and standard templates are used by traffic engineers for intersection design.
- The similar mechanical solutions applied in the design of all vehicles. For example, the gear-change positions are set to similar places in most vehicles.
- A very limited and universally-applied set of traffic rules.
- The natural tendency of a human driver to repeatedly use the same route, in order to avoid unexpected situations, which cause stress. For example, drivers will generally use the same route to work everyday and are very reluctant to change it, except when necessary.
- A person's driving is, in general case, limited to a relatively small number of places. A recent study has shown [Ashbrook, 2002] the number of locations

Figure 3.3. This image shows all GPS data captured by the user during a four month period [Ashbrook, 2002].

where someone spends more than ten minutes is very small. The subject of this experiment was equipped with the GPS for four months. During this time, the subject spent more than 10 minutes at only 25 locations and used only a small number of routes between them as could be seen in Figure 3.3.

There are a number of frequently occurring patterns in driving. On the higher abstraction level, it is very likely that the vast majority of everyday driving routes are to the same places (job, children's schools, favourite restaurant) using the same route. Repeating the known route reduces driving strain and is a logical choice for any driver. Any particular route can be represented as a sequence of driving events. We use the term *driving event* to refer to any major change in vehicle attitude or speed, such as a left or right turn, a stop, making a U-turn and so on. A similar pattern of driving events is repeated every time a driver uses the same route.

On the lower level of abstraction, each driving event also represents a pattern of simple driver actions. Turns, for example, consist of the following actions: the driver first reduces speed and changes to a lower gear, then rotates the steering wheel, next increases speed, changes to a higher gear, and finally, rotates the steering wheel in the other direction. This set of actions is repeated every time a driver executes a turn.

Further, these low-level actions (which a driver executes subconsciously) represent patterns of motion of the driver's limbs (and other body parts). Recent research [Grasso, 1998] showed that during normal locomotion around corners, gaze (head on trunk and eye in orbit) turned in the new direction with remarkable regularity. Pentland and Liu claimed that a driver's intentions could easily be predicted from the positions of his eyes [Pentland, 1995].

When learning to drive, drivers first learn to execute simple actions. They learn to coordinate their arms and legs in order to properly execute the most typical patterns of operation, such as changing gear and starting and stopping the vehicle. When a driver masters these simple operations, he/she takes on more complex patterns of action on a higher level of abstraction. A driver continues to learn patterns during his/her lifetime, constantly improving abilities and expanding experiences.

Unfortunately, due to the complexity and dynamics of the transportation system, a driver is unable to learn all patterns, and it is likely that he/she will forget patterns that occurred further in the past, as well as those that were not repeated often. For the same reason, in some situations a driver is not able to apply his knowledge in the required time frame. As has been elaborated in the previous chapter, a computer has a much faster processing speed than humans, and generally has a larger available memory for learning patterns. In this respect, we can conclude that a computer may have some advantages over humans in learning driving patterns.

We believe that by identifying and learning patterns in someone's driving, a computer-based system can provide information that can help a driver to fulfil a range of different navigational driving tasks. Learning from previous driver's experiences has many advantages over conventional systems. It allows adaptation of the system to a particular environment or to the individual driver's style. For example, a set of driving patterns for the same road could be very different in the morning than during the night, and for some purposes these may be treated as two different driving experiences. Consequently, the system should be able to differentiate between these two cases. On the other hand, two segments of a driving route on different geographic locations can be very similar from a guidance perspective, in which case the system may treat them as equivalent. Systems that are not based on learning techniques cannot identify and use such similarities.

We believe that drivers require help mostly for execution of medium-level navigation tasks. Medium-level driving tasks are also known as guidance, tactical, or intermediate. They involve selecting and executing manoeuvres to achieve short-term objectives, such as the maintenance of a safe speed and a proper path relative to the road and traffic elements [Lamm, 1999], resulting in a sequence of driving events. Low-level navigation tasks involve only a very small number of objects in close proximity, so their complexity is not high, and such systems are generally easy to implement.

On the other hand, high-level tasks, such as finding shortest or fastest route to the desired place, do not have hard real-time limitations. If the driver does not find the best possible solution for the task in the required time, no life-threatening situation will arise, but the chosen route will simply be suboptimal. Guidance navigation tasks have real-time constraints and deal with relatively complex and dynamic models, so their execution may, in extreme situations, overstretch a driver's capabilities. In such situations, computer-based systems based on learning driving patterns could be very useful.

A mathematical model of a driving pattern can be very simple. It consists of a sequence of vector states for a sequence of consecutive time epochs. If we denote driving patterns with Π and a vector state at epoch i as x_i, we may express driving patterns as: $\Pi = x_0, x_1, \ldots, x_n$. The vector state consists of a set of observed parameters of the traffic system. We will denote this vector state as: $x_i = \{a_i, b_i, \ldots, r_i\}$.

In the traffic model, learned driving patterns could be useful in a variety of ways. For example, we can extract the most common driving patterns and use them to improve driver education. However, for the traffic control loop we presented in Figure 3.1, the most appropriate use of learned patterns is to predict future driving events or required driver actions. This information could be used inside the control loop to augment the human driver in handling unexpected situations caused by random disturbances. If we can reliably predict most relevant aspects of future events and provide adequate response to perceptions, we may even replace the human driver by providing an automatic driving controller. With this in mind, in this thesis, we will characterise success in learning driving patterns by the system's performance in predicting future events.

The system for learning driving patterns must contain the following components:

a) The set of previously observed driving patterns, which we will call the *pattern history*. The pattern history can be stored explicitly or implicitly, but must preserve information required to generate desired output from the system (such as predicting future events). We denote pattern history as: $\aleph = \{\Pi_1, \Pi_2, \ldots, \Pi_j\}$.

b) An algorithm (function) to properly update the pattern history by presenting the data for the current vector state. This is the learning process, where sequences of events are identified and the pattern history updated to reflect the identified sequence. We denote the learning function as: $\Im = (a_t, b_t, \ldots, r_t) \rightarrow \aleph$

c) An algorithm (function) to predict future events from a pattern history (or to provide other useful information). This function is denoted as: $\Re = \aleph \rightarrow (a_{t+1}, b_{t+1}, \ldots, r_{t+1})$.

Machine learning techniques have already been used to successfully learn and predict temporal and spatial data in different domains. We believe that they could also be used to successfully learn driving patterns. In this thesis, we will argue that driving patterns can be successfully learned using two popular machine learning paradigms: neural networks and hidden Markov models. In the following chapters, we present the methods we developed for learning driving patterns based on these techniques.

3.3. Situation model – key to traffic decision making

The competence of human drivers in solving complex navigation problems in traffic comes from their situation awareness. Situation awareness is a task specific understanding of the current situation including position of the driven vehicle, other vehicles in the vicinity, and intentions of their drivers. Many research topics in intelligent transportation systems over the years could be considered to build specific aspects of systems with situation awareness. However, only recently have researchers explicitly declared the need for situation awareness in intelligent transportation systems [Sukthankar, 1997].

In the intelligent vehicle, the set of data that contain situation aware information creates the *situation model*. In order to make correct decisions while driving, the situation model must be comprehensive. It must include data about the current state and future intentions of the driver, his/her vehicle, and various objects in the environment. The model must also build and maintain relationships among these entities, as they are strongly correlated in traffic systems. These relationships can be stored in the situation model explicitly by storing data that describe the relationships between entities. They can also be stored implicitly, by gathering data that describe the common behaviour of a set of entities. Such data preserve not only actions of separate entities, but also their relationships.

As we previously stated, the decision-making process in driving has very strict time limits. For example, if a vehicle approaches an intersection, the driver must make a decision about his/her action at the intersection before he/she arrives there. Any failure to make a decision on time could easily cause an accident. Attempts to increase the available time for decision making by decreasing vehicle speed are rarely successful, due to the pressure imposed by drivers of other vehicles.

When the situation model is implemented, the strict time limit translates into requirements for compact data storage and fast processing. Many conventional computer science techniques for decision making are not applicable, as they use brute force search techniques. Artificial intelligence approaches are much better suited, as they are developed to work in complex domains. The fact that humans learn to drive and improve with experience suggests that an intelligent vehicle could also acquire situation awareness through repeated exposure to traffic situations. This clearly indicates that machine learning techniques are well suited for building situation models.

The situation model does not influence reactive navigation tasks, as they are direct responses to sensory perceptions. However, in order to solve higher level (guidance and planning) navigation tasks, a certain amount of situation awareness, represented as a subset of the situation model, is required as illustrated in the Figure 3.4.

The situation model is used in conjunction with the “world” model. The world model contains the knowledge about environment: maps of the road network, information about weather and traffic conditions, and so on. Generally, we may say that the situation model encapsulates dynamic aspects of the traffic system on the micro level

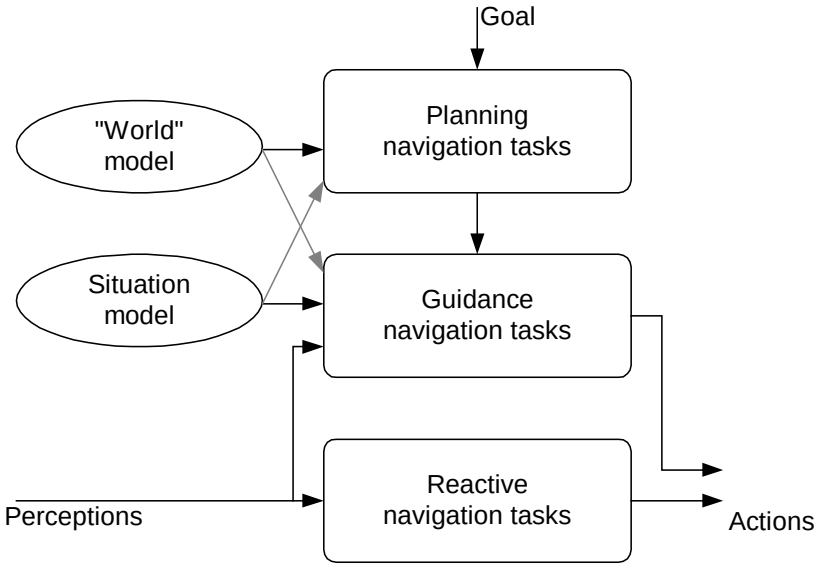

Figure 3.4. Situation model for solving navigation tasks.

(in the vicinity of the particular vehicle), while the world model encapsulates static aspects of the traffic system and also dynamic aspects on the macro (global) level.

Higher level navigation tasks generally require the more complete world model. However, they do not require the situation model to be as comprehensive, as it is related mostly to the local environment. Some planning tasks, such as finding the shortest path to a particular place, require a very simple situation model, which consists only of the position and the attitude of the vehicle.

The amount of data from the situation model used in a particular navigation task depends on the problem to be solved by this task, as shown in Figure 3.5. Some guidance tasks, like tactical driving at complex intersection, require a larger amount of information from the situation model. On the other hand, simpler tasks, like driving on the straight road segment, require a much smaller subset of data.

We firmly believe that pattern learning could be used very successfully in building a situation model. The values of the vector state in the current and a few recent time epochs represent information about a situation, while predictions, generated from pattern history and the current state, contain information about the intentions of traffic participants.

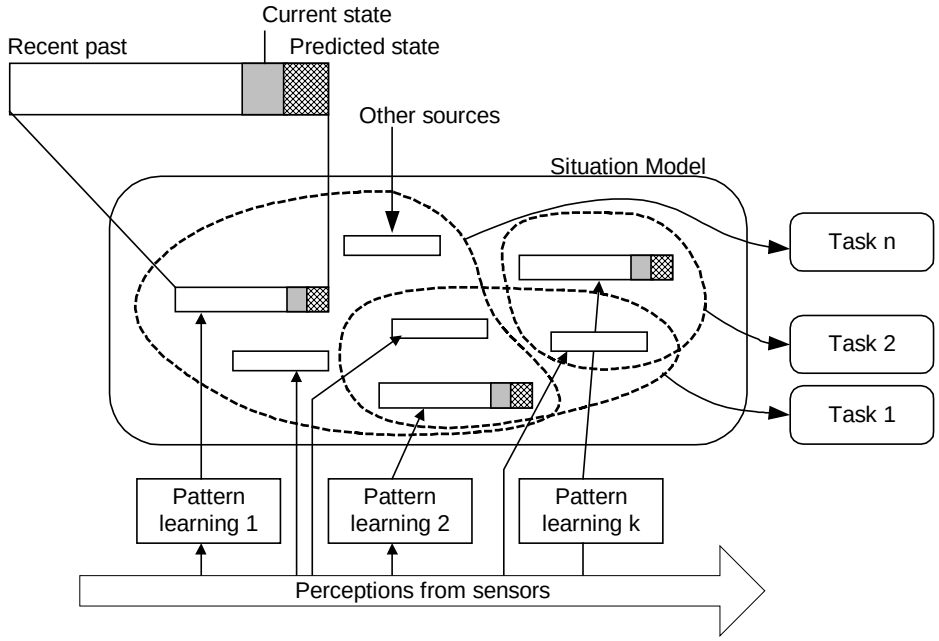

Figure 3.5. Learning driving patterns as a building block for the situation model.

It is obvious that a single system, regardless how advanced, will not be able to cope with the complexity of the traffic system. As was already experienced in the domain of autonomous mobile robotics, monolithic systems do not have the required flexibility and are not able to successfully solve navigation problems. It seems obvious that the solution must be in the form of an organised network of simple modules, each dealing with some aspect of the traffic system. One such distributed system for mobile navigation is proposed by Rosenblatt [Rosenblatt, 1997]. It is based on a number of behaviour-based modules such as road-following or obstacle avoidance, which operate in parallel. The modules send votes about further actions to the command arbiter, which makes the final decision. We believe that many modules can be implemented by using a driving pattern learning approach and that they could be further integrated using the approach proposed by Rosenblatt.

Most of the previous research in intelligent transportation systems resulted in complex methods and bulky systems. However, we believe that complexity and cost of modules must be kept at a minimum, because many modules will operate in parallel in a complex and dynamic domain. One of the goals of this thesis is to prove that a simple

and inexpensive system can provide useful information to the driver and increase the safety of driving.

Building a comprehensive situation model is out of the scope of this thesis. However, in our experiments described later, we will prove that our proposed techniques for learning driving patterns can provide accurate information about the current state of the traffic system and reliably predict the future events in the system. Information about the current and predicted state can be further used to build a situation model, as proposed above, though this will not be further elaborated.

3.4. The framework of the driving warning system

As we discussed in the previous sections, random disturbances cause traffic events that do not follow the usual patterns of events. Accidents happen when drivers do not react properly or fast enough to a changed situation. During evolution, human capabilities to react to external events were adapted to relatively slow events in nature. Modern cars are, however, much faster than any living creature on the Earth. For example, a car travelling at 100km/h will go 30 meters before the driver can react to some external event. This reaction time can be even longer depending on the driver's age, tiredness, and other factors.

People who drive cars tend to predict future actions of other traffic system participants and include these predictions in their decision-making process. However, humans tend to avoid thinking about unpleasant things and refuse to account for facts that do not fit into their perception of the situation. Consequently, when an unexpected event happens it is very hard for a driver to adapt.

On the other hand, it is known that computers process information much faster than the human brain in many domains. For simple problems, they are able to provide an almost instantaneous reaction to external inputs. Even in complex situations, like local traffic systems, a computer's reaction time will be much shorter than that of a human driver. Computers are not affected by tiredness and cannot be influenced by drugs or alcohol. They do not have preferences, and may regard all possible outcomes of the particular driving situation with the appropriate attention. Almost unlimited data storage

in modern computers enables even very old and rare driving experiences to be stored and later used in the decision making.

The advantages of the computer over human drivers motivated many research projects on *driver assistants* and *driver warning systems*. These systems are intended to either help a driver execute certain navigation tasks while driving, or to provide warnings when a dangerous situation is detected. The ProLab 2 system for driving assistance was developed as part of the European Prometheus research programme [Rombaut, 1993]. Its goal is to inform the driver about the potential or real risk involved in executing a given manoeuvre. It consists of multiple video cameras, image processing, and temporal multi-sensor fusion components, while multiple real-time expert systems are used for decision making. The University of German Armed Forces developed DAISY, a driver monitoring and warning system based on a aircraft pilot assistant system [Onken, 1994]. This system helps drivers in maintaining longitudinal and lateral control of their vehicles on German motorways. Warnings are generated from discrepancies between the model of the reference driver and the model of the actual driver.

Here we propose a framework of the driver warning system based on learning driving patterns. The goal of the framework is to warn the driver if/when the driver's current behaviour becomes significantly different from usual in the context of the routes he/she usually travels.

The system can be equipped with a number of sensors to measure values for those parameters of the driving system that we are interested in. The microscopic traffic system consisting of the driver, the vehicle, and the environment can be described by a large number of different parameters. The minimal set of parameters describing the current state of the system consists of the vehicle's speed, direction, and acceleration. These parameters are related to the vehicle; however, they implicitly contain information relevant to driver behaviour and traffic infrastructure, because they significantly influence the current state of the vehicle. Additional parameters that are related to the environment (such as current visibility and temperature, time of day, distance from the vehicle in front) and driver (his/her attention, age, and tiredness) could also be considered highly relevant to the state of the traffic system. Selected parameters can be measured by data acquisition hardware and software, and numerical values for

parameters are generated. These numerical values build a vector state x_t for our pattern learning system.

As we previously described, the learning part of the system uses input data to build and maintain the traffic system model. This model consists of the current state of the system, recent history of the system, and the probability (calculated by the system) that the model represents the actual state of the traffic system (how reliably the model represents the actual situation). The pattern learning module uses data from the pattern history to detect if the current state of the model represents an instance of previously encountered patterns. The probability that the current state represents a pattern from the previous experience is calculated by the system. The pattern learning module also updates the pattern history to reflect the current system experience.

The prediction module finds patterns from the pattern history that are similar to the pattern in the current model. This set of similar patterns is used by the prediction module to calculate the most likely future events. This prediction is the most likely next state for the traffic system, denoted here as p_t.

The predicted state can be compared with the actual state of the system at the same time, denoted by c_t. If the system follows the most probable pattern of events, the actual state should be the same as the predicted one. In many cases, the system will not follow the most probable pattern of events. However, due to the mechanical nature of the vehicle, the standard design of traffic infrastructure, and other reasons noted previously, the actual state will likely be similar to the predicted one. If the difference between the actual and predicted states is significant, then the specific pattern of events is not part of the pattern history. As such, the new pattern may signify a new experience or some uncommon situation, which may cause an accident. The number of patterns that are not in the history decreases very quickly with the amount of time the system is in use. After the pattern history contains data from most of driver's usual routes, the system will be able to detect uncommon situations. Comparing current and predicted states, and deciding whether to generate warning messages, is performed by the module we refer to as *State assessment*. The framework for the driver warning system based on learning driving patterns is presented in Figure 3.6.

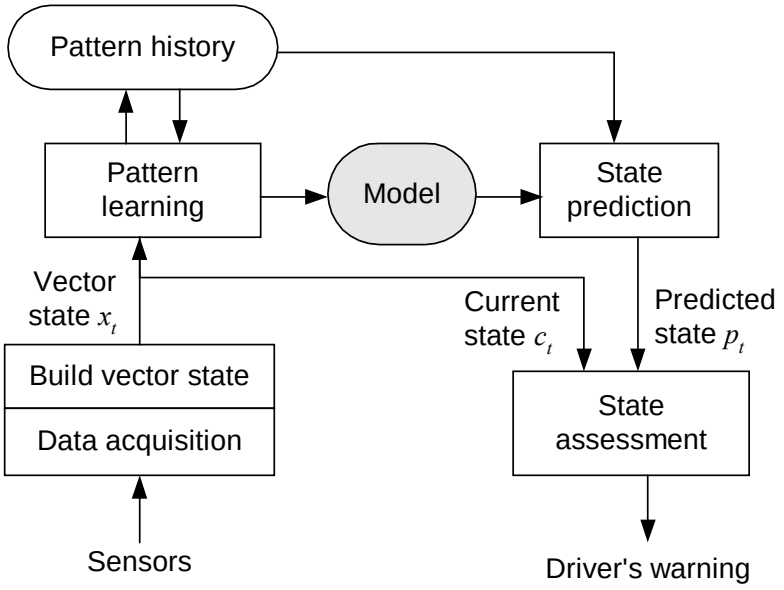

Figure 3.6. The framework of the driving warning system.

To be usable, the framework must operate in real-time. The eventual warning for the driver should be generated in a very short time – shorter than the time the driver requires to assess the situation by himself. Many machine learning techniques are able to generate predictions very quickly. However, learning is usually based on repeated exposure to the training data and requires more time than is available in real-time operation. Fortunately, learning does not have to be executed in real-time and can easily be organised in periods when the system is not operational (the vehicle is stationary, for example). A number of other techniques are developed over time to reduce learning requirements for machine-learning algorithms. In the future, similar systems may be able to take control of the car from the driver, when a high probability of a possible accident situation is detected.

It can be argued that the practical usability of the previously described system is not high. For example, it will not work for previously unencountered routes, it does not provide support for highway driving, and so on. We are aware of these limitations and state again that a system for comprehensive driver support would consist of many modules similar to this framework. However, we believe that, inside such a comprehensive system, support for a driver's usual routes must have an important place,

because they represent a large percentage of the travelled mileage. We also believe that this framework represents a simple but robust way to provide the required support for a specific navigation task.

The implementation of our proposed framework is outside the scope of this thesis. The framework is proposed here to provide an indication of the possible use of the methods for learning driving patterns developed in this thesis.

3.5. Other applications of driver pattern learning

The driving warning system is not the only possible application of the techniques for learning driving patterns. In this section we briefly present some further opportunities.

Digital map matching is a process of matching data from vehicle sensors to the digitised road map stored in the computer in order to recognise the vehicle's present location [Zhao, 1997]. This technique is often used to augment GPS-based navigation systems in "urban canyons" where GPS signal is not available, or to provide dead-reckoning-only navigation solutions.

We can use the system for learning driving patterns to reliably recognise driving events such as turns, intersections, and so on). If we know the starting point of the vehicle, we can use recognised events to track the vehicle's movement through the network of streets stored in the appropriate data structure. Even if the starting point is not known exactly, the system could be able (if the street network data structure is properly organized) to search a road network for a route that may generate such a sequence of events.

Vehicles that have fixed routes (buses and trams in local transportation) may use driving patterns as a sole means for finding location. Pattern history can be generated by a demonstrational drive and can serve as a "map" of the route. Comparing the current state with the pattern history will generate the position of the vehicle, relative to the demonstrational drive (predefined route). In such cases, it is very easy to detect deviations from the predefined route which can indicate a major problem in vehicle operation or some other unexpected event that may be significant for security reasons.

With advances in navigation sensors and mobile computing, research in personal navigation has become popular [Legat, 2001]. The goal of these systems is to enable the tracking of children and elderly people for safety reasons, or to enable navigation in an unknown environment (another city, in the forest, and so on). The patterns of someone's walk could be used in a manner similar to vehicles: to improve the accuracy of the location or to provide high-level monitoring. For example, the system could easily detect if a child does not follow the usual path to the school, and send a message over the mobile phone to inform his/her parents.

Tiredness is very often a cause of traffic accidents. A tired driver responds to sensory perceptions more slowly than in normal circumstances. As a result, a pattern of driver's actions, while tired, looks different from his usual patterns. We believe that it is possible to detect the moment when a driver's behaviour becomes too different from his/her usual behaviour, but before the driver and vehicle pass the safety limits and an accident occurs. A simple warning at that stage can easily save the life of the driver and/or other people. A system for detecting tiredness may have training patterns collected while the driver is in 'normal' condition. The measured difference between these patterns and patterns currently detected can indicate the level of driver's tiredness.

A similar approach could be used as a supporting tool for learning to drive. In this situation, the training data could be obtained by collecting driving data from experienced drivers. The system could be organised to learn which action to execute in order to follow the patterns provided by "expert" drivers. These "instructions" could then be presented to the novice driver, who will learn by mimicking the recorded activities. The interface for transferring instructions to the learner should be flexible, because different people have different preferences in learning. Some like to be guided, while others prefer to have feedback only in situations when a major mistake is made.

3.6. Related work

The investigation of driving patterns is a very popular topic in the traffic-related research community. Driving patterns contain important information about traffic systems that can be used for many different purposes.

Since the time of the great oil crisis in 1973, investigations in vehicle fuel consumption have become very important. Very early researchers came to the conclusion that patterns in the speed and acceleration profiles of a vehicle considerably affect fuel consumption. The New Zealand Liquid Fuels Trust Board financed experiments to detect driving patterns of New Zealand vehicles. The patterns were measured by selecting a random vehicle and following it in a car equipped with the required sensors [Lee, 1984]. These patterns were used to better understand the usual driver's behaviour and to propose changes in the traffic infrastructure to better suit drivers.

A very similar approach was used by [Austin, 1996]. A chase car followed light-duty vehicles over 102 different routes in the South Coast Air Basin in California. Uniform distance was maintained between the chase car and the subject car with a range finder laser developed for the study. The chase car collected data on chase car speed, manifold air pressure, and acceleration, subject vehicle speed and range, and other parameters. The data were broken into 833 microtrips (a microtrip is the portion of travel that occurs between stops) and assembled into a driving cycle representative of real-world trips of about 20 minutes' duration. A new speed-versus-time vehicle emissions driving cycle model for light- and medium-duty vehicles was developed, which more accurately reflects real-world driving patterns than did the existing standard test cycle. Emissions of carbon monoxide and oxides of nitrogen resulting from the new driving cycle were found to be higher than those predicted by an older cycle.

Recent research at Lund University in Sweden also deals with characterisation and environmental implications of urban driving patterns [Ericsson, 2001]. Human subjects drove cars equipped with sensors and data loggers. Various parameters related to the environment and driver behaviour were recorded. More than 19,000 driving patterns were described using up to 62 parameters. Some parameters, such as vehicle acceleration, power demand, engine speed, and gear change position, were found to have significant effects on emissions and fuel consumption. Researchers found that street configuration and driver characteristics affected driving patterns.

Modern cars have many components controlled by electronic devices. The first generation of electronic controllers generally mimicked the behaviour of mechanical devices. Due to a decrease in the price of microcontrollers, the newest generation of cars

have significant processing capabilities, and some manufacturers use them to provide device adaptation to the current situation. For such devices, driving patterns represent a significant source of data. Volkswagen's electronic automatic transmission device monitors driving patterns and accordingly changes the way gears are shifted [Volkswagen, 2001]. Heavy accelerations and frequent speed changes result in a sporty shift program, with delayed up-shifts and early down-shifts for superior performance. Steady speeds or mild accelerations, on the other hand, call up a fuel economy program, with early up-shifts and delayed down-shifts.

Adaptive cruise control systems control relative speed and a vehicle's distance from the vehicle in front of it. In order to design an adaptive cruise controller, a model of a driver's longitudinal behaviour (keeping distance to the vehicle in front of it) is required. Bengtsson compared several methods for building models of longitudinal behaviour, including neural networks, linear regression, and behaviour models [Bengtsson, 2001]. Two vehicles (Volvo 850 with automatic transmission) were used in a leading-vehicle-following-vehicle experimental setup. The following vehicle was equipped with radar and laser sensors to measure its distance to the leading vehicle. The collected data was used to build models of a driver's longitudinal behaviour. Bengtsson also investigated the use of the Generalised Auto-Regressive Conditional Heteroscedasticity (GARCH) method to detect changes in driver behaviour.

Researchers working to automate low-level (reactive) driving tasks (like line following and obstacle avoidance) very often use data from previous driving experiences. The ALVINN system, for example, uses the image of the road ahead to make decisions about the position of the steering wheel in order to maintain the vehicle on the road [Pomerleau, 1989]. The driving patterns represented as sequences of images are used to train a neural network. When trained, the neural network calculates the steering wheel position for the vehicle on the basis of the current image from a camera. The author claims that the car equipped with ALVINN autonomously drove thousands of kilometres on highways across the U.S.A. ALVINN is described in more detail in Chapter 5.

Yu and Sethi [Yu, 1995] proposed a road-following system that uses a combination of the neural network and reinforcement learning techniques. This system uses video cameras as the principal source of the input data. The improvement over

ALVINN is in additional reinforcement information that is generated from input images and represents a 'hint' to the vehicle to move to the centre of the road. Thus, continuous learning capability, similar to that of a human driver, becomes possible. The authors supported their claims by simulating a simpler version of the proposed system.

A fuzzy clustering method for characterising driving patterns was described in [Haskel, 2000]. Data collected from vehicle sensors are grouped into clusters, which are further merged into cluster sets. The resulting group of clusters characterise a particular driver. A similarity measure was introduced to quantify the degree to which new cluster sets "fit in" to a cluster group. Using this measure, it may be possible, according to the authors, to recognise who is driving the car, or the type of driver (aggressive, cautious, and so on), or to detect a drowsy driver or other abnormal situation when an unexpected deviation from the usual behaviour is detected.

A team of researchers from MIT Media Lab and the Nissan Cambridge Basic Research Centre conducted research in modelling driver behaviour for a number of years. They proposed the global framework of an augmented control system, which helps the driver execute vehicle navigation tasks. Within this framework, the driver is an integral part of the control system. Efficient operation of a control system requires knowledge of the driver's model, which is used to design other parts of the control system. Consequently, their research goal is to provide a driver model that is, more or less, independent of other components of the transportation system (vehicle and environment).

In their first paper, Pentland and colleagues present a short overview of possible driver models, that can be used in an augmented control system [Pentland, 1995]. They found that static models (such as finite state machines) and simple dynamic models (such as processes that can be handled by Kalman filters) are not appropriate for traffic system complexity. Their approach is to model the human as a Markov device with a (possibly large) number of internal mental states.

To implement this, they used a so-called multiple model (or generalised likelihood) approach. Observations about a driver's state are made, and, based on these observations, a selection of an appropriate model is made. The proposed system will have a number of predefined dynamic models that could be used in various states. System response is then calculated, based on the selected model and input data. Kalman

filters are used for realisation of dynamic models. Hidden Markov models are proposed as a solution for the recognition of a driver's state. Results of the visual recognition of American Sign Language were used to demonstrate the feasibility of the proposed method [Pentland, 1995].

In a later paper [Liu, 1997] the same theoretical foundation was repeated, but with a focus on the recognition of a driver's intended actions. Experiments were conducted on a Nissan 240SX convertible driving simulator. The simulator was equipped with a number of sensors to record the driver's commands on the steering wheel, brakes, and so on. The simulator had a computerised control system to calculate the state of the dynamic car model (such as car velocity, acceleration, and so on). A wide-angle image projector was used to provide the user with a view of the environment in which he/she should drive the car. The same display was used to present the user with text commands about required driving actions to be taken.

In experiments with real-time recognition of driving intentions, eight male subjects were asked to "drive" the simulator in the city-like environment for approximately five minutes. The position and the velocity of the steering wheel were recorded with 10Hz frequency. The calculated car velocity and acceleration were also recorded with the same frequency. While driving, the users were asked to execute the following actions: (a) stop at an intersection, (b) turn left at an intersection, (c) turn right at an intersection, (d) change lanes, (e) pass the car in front, and (f) follow the car in front. A textual command (on a simulator screen) was presented to the driver to ask for a specified action.

Hidden Markov models were created for each driving action by using commercially available HTK software. The initial model in each case was a three-state model with uniform probability, which was later refined by applying collected data. The observation sequence consisted of collected data for steering angle and velocity, as well as car velocity and acceleration.

The goal of the experiment was to detect and classify a driver's action, a short time after presenting a command. Recognition results were tabulated at one, two, and three seconds after presenting a command to the driver. Models were trained on seven subjects and tested on eight. After testing models on each of the subjects, the results were averaged. The recognition rates were 88.3%, 89.4% and 87.5%, for one, two and

three seconds respectively. According to the authors, "these results demonstrate that many types of driving behaviour are sufficiently stereotyped that they are reliably recognized from observation of the driver's initial movements". The same authors reported similar results in [Pentland, 1999].

A further paper from the same team [Kuge, 2000] describes experiments for continuous recognition of lane changes. In this case, drivers in the simulator were asked to change lanes (to the next lane on the right) or to stay in their current lane. In some cases they were forced to execute an emergency lane change due to an obstacle in the lane (in the form of a large stationary truck). The steering angle, steering angle velocity, and steering force were measured from the driving simulator. Hidden Markov models were used to model driving events; for a normal lane change, a set of three models with three states in each model were created. For emergency lane changes the same architecture was chosen, while staying in one lane was modelled by a single model (also with three states). The authors reported recognition rate of 98.3% for emergency lane changes. Successful recognition was possible after 0.6 seconds and reached its maximum one second after the command presentation.

In further research on the same topic, the SmartCar testbed platform was developed in MIT's Media Lab [Oliver, 2000]. The SmartCar platform consists of a real-time data-acquisition system and a machine-learning framework for recognising driver manoeuvres at a tactical level. A Volvo V70XC was equipped with a number of sensors: speedometer, acceleration throttle, steering wheel angle sensor, brake pedal sensor, and GPS unit. External video cameras were used to record frontal and rear traffic. Two internal video cameras recorded the driver's face and his/her viewpoint via a camera mounted on the driver's glasses. Video signals from the cameras were combined and recorded on a video tape. Hidden Markov models were used in experiments and coupled hidden Markov models (CHMM) were proposed for modelling two interacting processes (the driver and the car).

The goal of the SmartCar experiment was to predict a driver's intentions after presenting a command, but before any significant change in vehicle position was detected. This work extends Pentland's and Liu's by modelling a larger number of manoeuvres (seven), using data collected on a real car (not a simulator), and including information about the car's position in the lane and the driver's gaze (from the video

source). The results of the experiments showed that most manoeuvres at the tactical level can be predicted before any significant change in vehicle position, and that in some cases information about the car's position relative to lane and driver's gaze were important for recognition. However, based on the numerical results presented, we may conclude that the system was inconsistent – for example, it had a very low recognition rate for turning left – which may be caused by the complexity of the data supplied to hidden Markov models. The authors acknowledge that some manoeuvres (such as line changing) were not successfully predicted.

Fraile and Maybank also used hidden Markov models to learn driving patterns [Fraile, 1998]. Their system used image sequences of a car moving in a car park taken by a video camera. The geometric models of the car and car park were known. Image processing algorithms were developed to extract the position of the centre of the vehicle's rear axle. The sequence of measurements for each trajectory was divided into overlapping segments. Each segment was assigned one of four categories: ahead, left, right or stop. Vehicle trajectory was such reduced to the string of symbols from the set {a, l, r, s}. The authors used a four state hidden Markov model to correspond to these four categories. The authors did not provide any numerical quantification of their experiments, but reported that the proposed method provides for reliable trajectory classification and identification of abnormal behaviour.

3.7. Conclusions

Driving patterns are sequences of events in the traffic system repeating over time. They are caused by regularities in traffic infrastructure, vehicle design, and traffic rules, as well as the natural tendency of a human driver to repeat known events rather than to experiment.

We believe that by identifying and learning patterns in someone's driving, we can provide useful information that can help the driver to fulfil a range of different navigational tasks. Learning from previous drivers' experiences has many advantages over conventional systems.

In this chapter, we identified reasons causing patterns in driving on various levels of abstraction. We derived a simple mathematical model for driving patterns and described a methodology for generating predictions from learned driving patterns. The importance of the comprehensive situation model for the intelligent transportation systems was evaluated. We proposed a use of driving patterns to help build situation models.

The framework of the driving warning system, based on pattern learning was proposed. The framework will learn driving patterns and will be able to predict future driving events. From the difference between actual and predicted events, system should be able to detect a potential accident situation and warn the driver. Several other possible applications of driving patterns are also presented.

Finally we summarized the results of several previous relevant research projects. Some of these projects were very similar to the experiments we conducted. However, we believe that our approach has some advantages. Firstly, we propose using learned driving patterns to build a situation model and then solve multiple driving navigation tasks with it. Other papers limited the application of the learned patterns to a specific navigation task. Furthermore, one significant aspect of our proposed solution is simplicity, while most other research generates very complex solutions, which are hard to implement. In most cases, other systems are a result of larger research teams with many resources available. Our proposed method requires only a minimal set of inexpensive navigation sensors and off-the-shelf computing platforms, and uses well-known machine learning techniques.

4. Data Acquisition System

The goal of this chapter is to present the design and implementation of the data acquisition system. The system consists of a mobile computer, various navigation sensors, an analogue to digital conversion card, and data collection software. Each of these components is described in this chapter in more detail. Furthermore, we describe the limitations that significantly influenced our system design.

An extensive testing procedure was used to validate the data acquisition system after each development step, and it is described in this chapter. Data averaging and digital filtering were implemented to improve the collected data by reducing noise. The data acquisition software was designed and implemented by using object oriented techniques.

4.1. Introduction

In the previous chapter, we discussed the large number of different parameters that influence a driver and a vehicle during driving. The construction of a comprehensive driving support system requires the use of data from many different sources. However, building systems that have many input parameters is very difficult and often ineffective, even when large human and material resources are dedicated to them, as in the case of the European research project PROMETHEUS [Catling, 1990].

In the previous chapter, we explained that our goal is to demonstrate that even a very simple system based on several sensors may provide useful navigational support for the driver. When required, learned driving patterns can be integrated on a higher level with additional information. The advantage of this approach is that components that extract useful information from vehicle data can be simple, and that the number of sensors can be limited.

The resources available for this research were severely limited. We had only NZ$2000 available for all hardware and software, which is less than the average price of one sensor used by other teams working on similar projects. Furthermore, every result presented in this thesis is the product of one person's work, while in similar projects, teams of people work on different aspects. These limited resources were a major factor influencing the selection of sensors, the design and implementation of the data acquisition system, and many other solutions presented in this Chapter.

Other important goals in the data acquisition system design were to provide for easy software development and use of development tools that the author was familiar with, and to use off-the-shelf components whenever possible in order to reduce the cost and increase the flexibility and extensibility of the system.

4.2. The data acquisition system design

To collect data while someone is driving a vehicle, we needed sensors and a mobile computing platform appropriately equipped to collect data from the sensors. The output

from most navigation sensors is still in the form of analogue signals, with a voltage or frequency proportional to the value of the physical phenomena that is measured. However, the algorithms for pattern learning require input data in the form of digital numerical values in a predefined range. In order to convert analogue signals into digital numerical values, a special hardware device for analogue-to-digital (A/D) conversion is required. Furthermore, to experiment with pattern learning we required a computing platform that enabled interactive operation and had sufficient computing power.

These three requirements (mobile computing platform, A/D conversion, and sufficient computing power) were the starting points in the design of the data acquisition system. The Computer Science Department provided a powerful desktop computer, so we focused on the mobile part of the system. The architecture of the solution composed of two separate computing platforms, one mobile and the other in the office, is illustrated in Figure 4.1.

The following options for a mobile computing platform and A/D conversion component were available when we selected components (1997):

- Controllers for embedded applications based on 8-bit and 16-bit micro-controllers with embedded A/D capabilities;
- Single board computers based on Intel Pentium class processors. Some of them are implemented with Peripheral Component Interconnect (PCI) local bus master control. Such solution enables the use of the passive PCI backplane, through

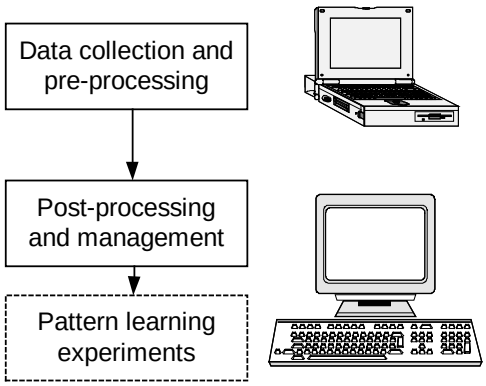

Figure 4.1. Two computing platforms for experiments.

which additional PCI cards, including A/D cards, can be connected. Other single board computers use a PC-104 bus, which is popular in automation and control applications.

- Standard laptop computers equipped with PCMCIA (Personal Computer Memory Card International Association) interfaces through which appropriate A/D conversion hardware can be connected.

Unfortunately, prices for single board computers and data acquisition PCMCIA cards were high and these options were more costly than our available research funds. On the other hand, the prices for a micro-controller based system were acceptable. However, this solution was incompatible with advanced software development tools, which are not available for micro-controller platforms. Writing the data acquisition software in an unfamiliar software development environment for untried hardware can take a very long time.

As none of these solutions was appropriate, we decided upon an unorthodox resolution: to build the mobile computer from standard parts used for desktop computers. In the second half of the 1990s, the PCI bus rapidly became standard for PC expansion cards, due to the much higher bandwidth required for computer graphics cards and access to data on magnetic and optical media. The prices of previous generation Industry Standard Architecture (ISA) local bus products were slashed as these lines were discontinued. Prices of ISA A/D conversion cards, which can collect data from several sensors, with selectable voltage range and conversion speed of at least 10^4 samples per second, were well within our budget. After some investigation, we selected NuDAQ ACL-8112DG, an enhanced multi-function data acquisition card, described in more detail later in this chapter.

The first version of the mobile computer was assembled from a second-hand motherboard based on a Intel 80486 processor and 200MB hard drive (parts of central unit purchased for $60) and a small PC case from an old departmental AT-compatible 80286 computer. The main problem with using a desktop computer in a car was to provide the appropriate power. One option was to purchase a special PC power supply component that uses car-battery 12V DC as the power source. However, these power supplies were rare and expensive at the time, so we decided to use the existing (220V AC) power supply component already mounted in the selected case. To provide 220V

AC from the car battery, a standard and inexpensive power inverter rated 150W was used. Solutions based on power inverters are known to cause problems when used to power computers; however, we did not experience any problems during experiments and found this solution to be reliable.

Due to the slow processor in our original motherboard and a small hard drive, we were forced to use the Microsoft Windows 3.1 operating system. This solution worked well and we were able to collect the first data sets. However, the use of the 16-bit Windows 3.1 constrained us to the Microsoft C 7.0 software development system, which is far more limited than later versions known as Visual C++. In order to overcome the limitations, we expanded the system with a new motherboard with Pentium processor and a larger hard drive. We also added a CD drive for easier software installation, and an audio card for improving the user interface; as will be explained later in this chapter. We were able to install and run the Microsoft Windows 95 operating system on the improved configuration.

General-purpose multitasking operating systems are not recommended for implementation of real-time systems, because they cannot guarantee a fixed response time. However, our data collection system was very simple, without any hard real-time requirements, and no other tasks to run in parallel. On the other hand, the mobile computer with the new processor had much more processing power than that required for the data collection task, making the complete system fully appropriate for the data acquisition. The use of a dedicated real-time operating system would have been possible, but the penalty would have been a long learning curve for software development.

The selected solution for the mobile computer platform is unorthodox, but over time proved fully usable and reliable. We were able to develop data acquisition software on the powerful desktop computer and then test it, debug it, and make minor changes on the mobile computer. In this way, we were able to save human and financial resources, and to use them in more creative ways.

4.3. Navigational sensors

The traffic system consisting of the driver, the car, and the environment can be characterised by a large amount of diverse information. The driver and his behaviour can be described by age, experience, tiredness, vision, the position of his head and the gaze of his eyes, and the force he applies to the steering wheel, brake pedals, and so on. The vehicle and its current status can be described by position, speed and acceleration (in all three dimensions). We may also measure the present state of the engine, differential, and many other vehicle subsystems. For each of the objects in the environment, position, size, visibility and many other properties can be used. We may also use relative distances and speeds, such as the distance between the vehicle tire and the road edge, or the relative speed to the vehicle in front, and so on.

The comprehensive driver support system, as described in the previous chapter, should include as many information sources as possible in order to generate as accurate a situation model as possible. However, the complexity of the system increases exponentially with the number of driving parameters used. Because the goal of our research is not to develop a complete system, but rather to investigate the possibilities of learning driving patterns, and due to limited human and material resources for our research, in our experiments we used only a limited number of sensors.

The main goal of this research is to learn and predict patterns of driving events, which represent vehicle motion. For this reason, we chose a minimal set of information to characterise the vehicle motion, consisting of vehicle speed; longitudinal, lateral, and vertical acceleration; and the vehicle turning rate (again in three dimensions). It would be also useful to combine this information with sensors which would characterize the environment by measuring ambient light, temperature, humidity, etc. However, their use would significantly complicate already complex experiments.

Due to massive use of air-bags in modern cars, the price of accelerometers dropped significantly and made them readily available and easy for integration. To some degree the same applies to “solid-state” gyroscopes which are widely used for image stabilisation in cameras. We purchased accelerometers and gyroscopes and conducted preliminary tests. We found that, due to the flat nature of northwest Christchurch, where we planned to conduct experiments. In our preliminary tests, vertical acceleration was

negligible and we decided not to measure this in our experiments. Furthermore, collected data showed that the sensitivity of the acquired gyroscopes (Murata ENC-05D) was not appropriate to measure vehicle turns. Voltages measured while the vehicle was turning were under the sensor's noise level. Prices for gyroscopes with better sensitivity were much higher and outside our research budget. However, from the results of the preliminary tests, we found that we could use the measured lateral acceleration to detect even minor vehicle turns. Consequently, we decided not to use gyroscopes but to rely solely on acceleration data. The results of experiments presented later in this thesis confirmed the validity of our decision.

The vehicle's velocity was measured easily using signals provided by the vehicle's odometer sensor. For each wheel revolution, the sensor generates one electrical impulse. The number of impulses multiplied by the wheel circumfluence provides the distance travelled by the vehicle. The speed was easily calculated either by dividing the distance with the time spent for travel, or, more directly, by measuring the duration between two electrical impulses. Both approaches for speed measurement were easily implemented with the selected A/D card; however, this would have required changes in the vehicle's electrical connections. Because we used a private vehicle for our experiments, we decided to avoid any change in the vehicle's electronics. Consequently, we decided to use a GPS receiver to measure vehicle speed.

4.3.1 Micromachined uniaxial accelerometer ADXL05

Advances in the manufacturing of semiconductor electronic components in the last ten years have created a new class of electronic devices – micromachined integrated circuits (IC). They integrate microscopic mechanical moving structures with standard components of integrated circuits (analogue and digital electronic circuits) on the surface of a single chip. As outlined in section 2.2., the basic principle of an accelerometer's operation is the existence of a body of relatively large mass, with its position inside the sensor measured by some electromechanical phenomena. In micromachined accelerometers, a body of relatively large mass is created from the same piece of silicon as the rest of the circuit, using photolithographic micromachining techniques.

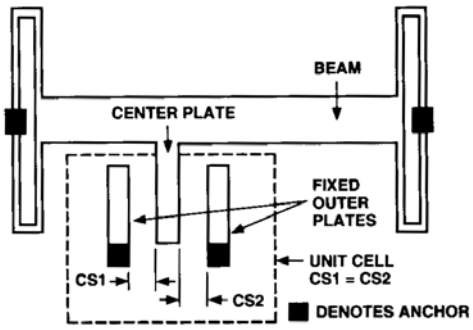

Figure 4.2. A simplified diagram of the ADXL05 sensor at rest [Analog, 1996].

As a pioneer in the micromachined IC industry, Analog Devices company shipped the first fully-integrated, single chip micromachining-based accelerometers in 1991. We selected an Analog Device accelerometer, model ADXL05 – a complete acceleration measurement system on a single monolithic IC [Analog, 1996]. It contains a polysilicon surface micromachined sensor and signal conditioning circuitry that implements a force-balance control loop. The ADXL05 is capable of measuring both positive and negative uniaxial acceleration to a maximum level of ±5 *g* (Earth's gravitational acceleration).

Figure 4.2. is a simplified view of the ADXL05's acceleration sensor at rest. The mechanical structure of the sensor consists of a common beam and 46 unit cells. Each unit cell consists of two fixed plates and the central plate attached to the common beam that moves in response to an applied acceleration. A movable central plate with fixed outer plates is, in effect, a capacitor. In this way, each cell unit is the differential capacitor (capacitive divider) sensor, consisting of two capacitors that are connected in series. The sensor's fixed capacitor plates are driven differentially by 1 MHz square wave signals. The amplitudes of these two square wave signals are equal, but are 180° out of phase with one another. When at rest, the values of the two capacitors are the same, and therefore, the total voltage output at their electrical center (i.e., at the center plate) is zero (the sum of two square wave signals in opposite phases is zero).

Figure 4.3 shows the sensor responding to an applied acceleration. In this case, the common central plate or `beam' moves closer to one of the fixed plates while moving further from the other. This creates a mismatch in the two capacitances. Consequently, one square wave signal influences voltage at the central plate more than the other,

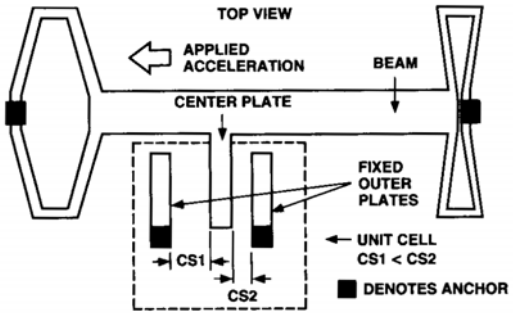

Figure 4.3. The ADXL05 accelerometer responding to an externally applied acceleration [Analog, 1996].

resulting in an output voltage at the centre plate. The amplitude of the output signal varies directly with the amount of acceleration experienced by the sensor.

Figure 4.4. shows an electrical block diagram of the ADXL05 sensor. On the left side of the circuit, the 1 MHz oscillator generates two square waves in opposite phases. These signals are applied to the fixed plates of the cell unit presented in the middle of the schema. For clarity, only one cell unit is presented, although the sensor actually contains an array of cells that are connected in parallel. The voltage output from the central plate of the sensor is buffered and then applied to a synchronous demodulator that is clocked, in phase, with the same oscillator that drives the fixed plates of the sensor. If the resulting voltage is synchronous and in phase with the clock, the result is a positive output. If the resulting voltage is synchronous but 180° out of phase with the

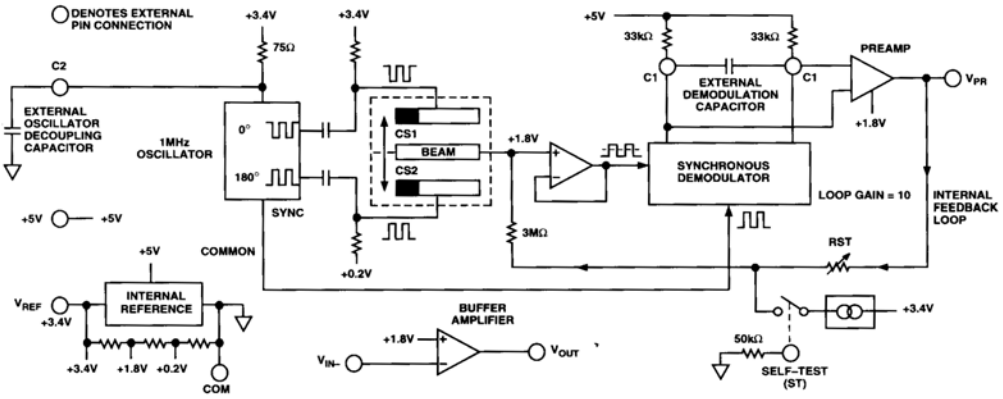

Figure 4.4. ADXL05 electronics -functional block diagram [Analog, 1996].

clock, then the demodulator's output is negative. All other signals are rejected. An external capacitor, C1, sets the bandwidth of the demodulator. The demodulator's output is amplified by a buffer amplifier. The resulting signal is available at the output pin V_{PR} and also fed back to the sensor to hold the sensor's centre plate in the 0 *g* position.

The ADXL05 sensor has another buffer amplifier, which can be coupled to the output (V_{PR}) pin. It can be used to set up a 0 *g* voltage level and the output sensitivity of the complete circuit, as well as to calibrate the sensor. To provide this additional functionality, a small number of external components are necessary [Kitchin, 1995].

Analog Devices also provides an evaluation board ADXLEB, a multipurpose printed circuit board, which simplifies the evaluation of the ADXL05 accelerometers. The evaluation board comes with a selection of external components that are required to set up an accelerometer in a variety of different configurations. The board itself enables these components to be soldered together with the accelerometer. After all components are in place, the complete board can be regarded as a very simple sensor with only three connections: a +5V power supply, and ground and output voltage proportional to the acceleration. In order to simplify our work (the design of the sensor board), we decided to use evaluation boards. We used the services of the Electrical and Electronic Engineering Department workshop for the integration of evaluation boards, as they required soldering of surface mounting components, which is not possible without special equipment.

Accelerometers have been set by external components to work with DC coupled connection with a full scale of ±2 *g*. A zero *g* bias level should produce DC output of 2.5V. The device sensitivity is set to 400 mV/*g,* which makes the total output voltage range from 1.7V to 3.3V.

4.3.2 Trimble Lasen SK8 GPS receiver

The Global Positioning System (GPS) was developed by the United States Department of Defense (US DoD) in order to provide precise positioning and navigation for various military units and vehicles, as described in Chapter 2. The high reliability and unprecedented quality of the positioning information for civilian use (even though it was until recently intentionally degraded by the US DoD) has made GPS the most popular

navigational device ever. There are a number of books [Hofmann, 1997] [Parkinson, 1996] and papers [Mitrovic, 1995] describing GPS and its applications in detail. In this section we provide only a brief description of GPS basic concepts and information specific to the receiver we selected.

GPS is based on a system of 24 satellites (the actual number of satellites is larger to increase system reliability) arranged in six orbits around Earth. Satellites broadcast two different pseudo-random codes and the "navigation message" in two frequencies in L-band (1.57542 and 1.22760 GHz). Pseudo-random codes are generated in shift registers, in which the value added at the end of the register is obtained as the result of a mathematical operation over two previously chosen cells. Each satellite is assigned a pseudo-random number (PRN), which determines the initial state of the shift registers and the order numbers of the cells on which the new value is calculated. A coarse Acquisition (C/A) code is dedicated for general use and the generated code is publicly available. The length of C/A code is 1023 bits and repeats every millisecond. Precision (P) code is a combination of two codes with a code length of approximately $2.3547 \cdot 10^{14}$ bits, with repeating time equal to 266.4 days. P code is not publicly known, as it is reserved for military use. The navigation message consists of 1500 bits of data containing patterns for synchronisation and diagnostics, data about satellites orbits (in two different precisions), numerical models for satellite clocks, information about the state of ionosphere, and a great deal of space reserved for military use.

Each satellite has high-precision atomic clocks (usually two cesium and two rubidium clocks in each satellite) as a time reference for signal generation and synchronisation. Because the signals have a spread spectrum nature, they are broadcast with relatively low power and on the same centre frequency for all satellites, which makes implementation of GPS receivers much easier.

GPS receivers consists of three basic components: a radio frequency (RF) converter, a digital signal processor, and a navigational calculator that is the controlling part of the entire receiver. The receiver can process signals from a large number of satellites simultaneously – modern receivers usually have 8 or 12 channels. The receiver internally generates codes (C/A for commercial receivers, P code for military receivers) using the same algorithm as the satellite. This signal is passed through a delay circuit, where the delay is tuned in order to obtain the high level of correlation of generated

signal with the actual satellite signal. If the receiver and satellite clocks are synchronised, the delay for the highest correlation is equal to the time it takes for the signal to travel from the satellite to the antenna phase centre. Using measured delay, the receiver can easily calculate the distance between the satellite and the antenna. Using distances from four satellites and their precise positions (received from the satellites as part of the navigation message), it is possible to precisely synchronise the receiver clock with atomic clocks aboard the satellites, and to calculate the receiver's position on Earth using the following formulas:

$$(X_1 + U_x)^2 + (Y_1 + U_y)^2 + (Z_1 + U_z)^2 = (R_1 + C_b)^2$$

$$(X_2 + U_x)^2 + (Y_2 + U_y)^2 + (Z_2 + U_z)^2 = (R_2 + C_b)^2$$

$$(X_3 + U_x)^2 + (Y_3 + U_y)^2 + (Z_3 + U_z)^2 = (R_3 + C_b)^2$$

$$(X_4 + U_x)^2 + (Y_4 + U_y)^2 + (Z_4 + U_z)^2 = (R_4 + C_b)^2$$

Here, X_i, Y_i, and Z_i denote position coordinates of satellite *i* (received as part of the navigation message), R_i is the measured distance between satellite *i* and the receiver (called the *pseudorange*). U_x, U_y, and U_z are receiver position coordinates, and C_b is the receiver clock bias (the difference between the receiver's clock time and the atomic time), as shown in Figure 4.5. The receivers can also measure the phase of the carrier

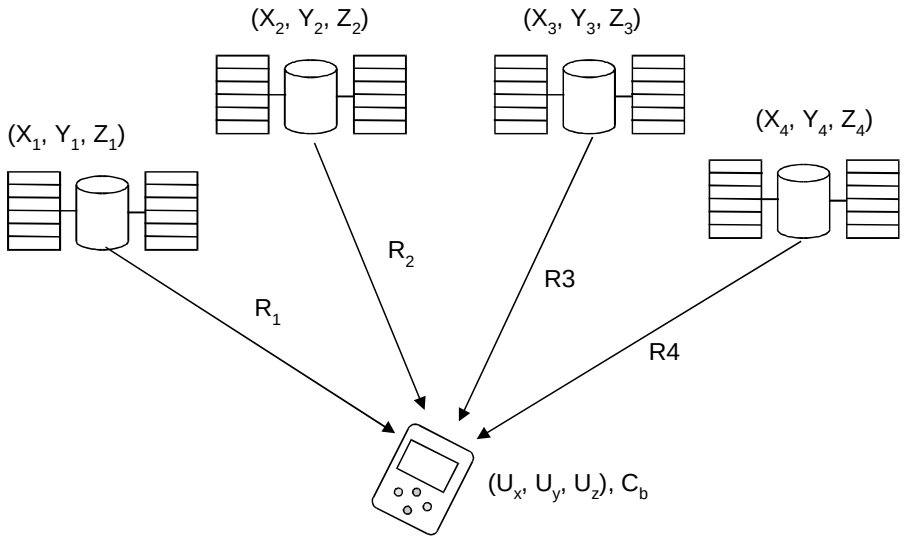

Figure 4.5. Signals from four satellites are enough
for the receiver to calculate its position.

signals, and the Doppler shift in the signals (caused by the velocity difference between the satellite and antenna). Carrier phase measurements can provide a higher level of accuracy of the position solution, but require sophisticated algorithms and a longer occupation of one position in order to solve so–called integer ambiguity (because the number of wavelengths between the start of the receiver generated carrier and the signal from the satellite is not known). The Doppler measurements cannot be used directly in position calculations, but can help to improve precision of the calculated position.

The accuracy of the determined position of the GPS receiver depends on the number of errors that occur when signals are generated on satellites and during their transmission, as well as while the signal manipulation in the receiver is in progress. Some of these errors can be cancelled by using additional calculations. For example, the ionosphere refraction error (which can be up to 30 m vertically) is frequency dependent, and can be compensated for by using signals from both carrier frequencies.

Differential GPS is based on the fact that the magnitude of error for determining the distance from the satellite to the receiver is approximately equal for all receivers that are in a relatively large area (for example, in a circle with a 200 km radius). If the receiver is positioned on a point with known coordinates, it may precisely calculate the actual distance from each satellite. By subtracting the actual and the measured pseudoranges, the receiver can calculate a *pseudorange error*. This calculated error can be used to correct measured pseudoranges in all receivers in a given area. This correction is called *differential* and can be applied either in real-time or by post-processing collected data.

The most important source of error is Selective Availability (SA) – deliberately introduced errors in the satellite signal (by the owner of the system, the US DoD) by which inaccuracies to the order of 100 meters are inserted into the C/A signal. Recently, SA has been discontinued and the accuracy of the horizontal positioning greatly improved. Currently, stand-alone receivers have an error of approximately 15-25 m, while differentially corrected positions have an accuracy of 1-3 m. If the carrier phase signal is used to calculate the position, an accuracy of approximately 10 cm can be obtained, but the receiver must be stationary while solving the carrier phase integer ambiguity. Relative measurements using a Real-Time-Kinematics (RTK) approach may have accuracy up to 1cm.

GPS receivers also measure the velocity of a vehicle or the platform on which the receiver is mounted. The simplest approach for measuring velocity is to use the position difference in two consecutive position measurements. The problem here is that positions can change significantly due to (for example) constellation change which could cause large errors in calculated velocity. Most modern receivers (including the one we selected) measure velocity by using the Doppler shift estimated by a receiver's search function. The speed of satellites is broadcasted as a part of ephemeris data which are collected by a receiver. The receiver can then accurately calculate its speed by applying the Doppler shift on a satellite speed. The measured speed can be improved by using differential corrections, and for a very precise speed measurements, by using a carrier-phase derived Doppler.

In our data acquisition system, we used Trimble Navigation's Lassen SK8 GPS receiver – a miniature 8-channel, C/A code continuous tracking receiver [Trimble, 1997]. Its high performance and low power consumption make it an ideal GPS engine for in-car navigation and telematics systems. The internal architecture of the Lassen SK8 receiver is illustrated in Figure 4.6. Like most of the current GPS receivers, it has two major custom-designed integrated circuits: one analogue (SCOTT) and the other digital (SCORPION). The analogue component converts radio frequencies (RF) of GPS signals to intermediate frequencies (IF), which are easier to process. The digital integrated circuit consists of the digital signal processor (DSP), which tracks GPS satellite signals from the highest eight satellites above the horizon, and an integrated 32-bit microprocessor (called a navigation processor), which controls the operation of the tracking channels, selects the optimum satellite constellation, and computes the position.

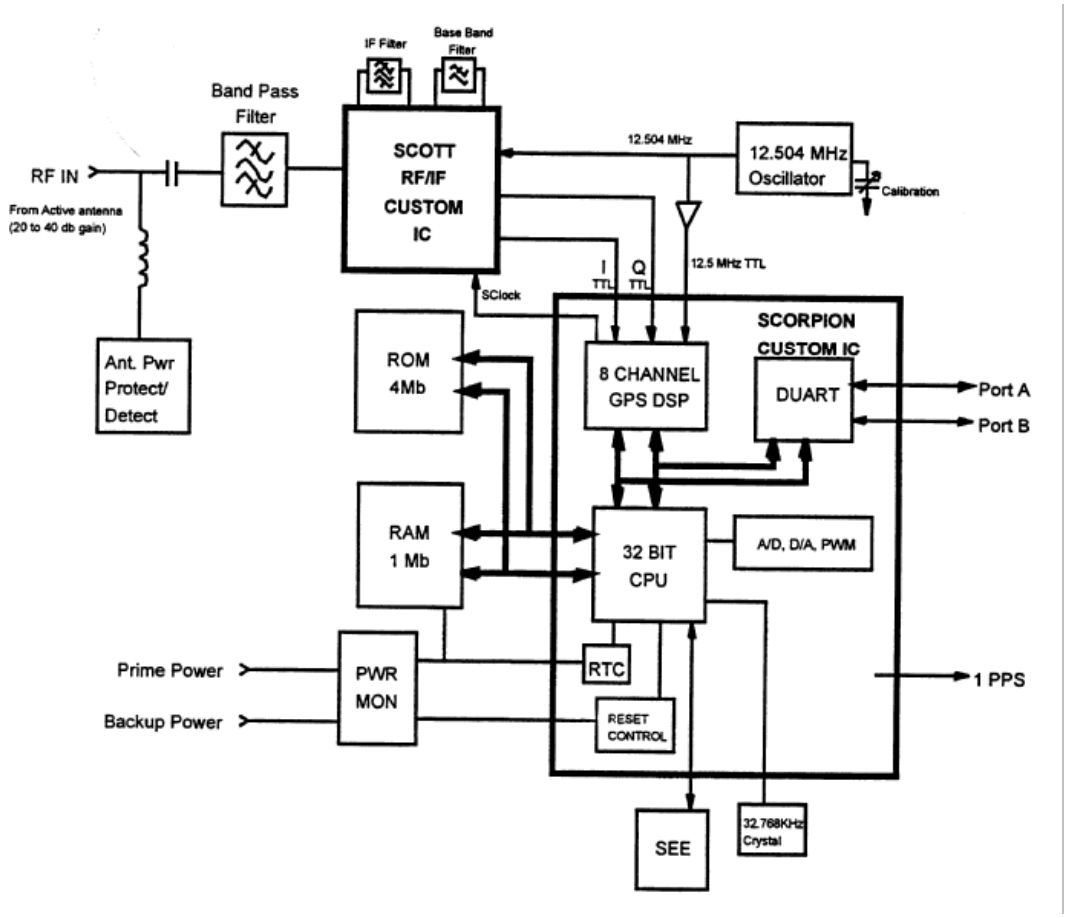

Figure 4.6. The internal structure of the Lassen SK8 GPS receiver.

The navigation processor also manages data from navigation messages and performs the data input and output. The Lassen SK8 receiver has two serial ports: one for data output, and the other for the input of differential corrections. It can output data in one of three different formats: TSIP (Trimble Standard Interface Protocol), TAIP (Trimble ASCII Interface Protocol), and NMEA (National Marine Electronics Association 0183 specification), and recognises RTCM (Radio Technical Commission for Maritime Services standard SC-104) format for differential correction messages. The receiver has an update rate of 1 Hz (it calculates a new position each second), a position accuracy of 25 m (50% of the time, without SA) or 2 m (with differential corrections), and a velocity measurement accuracy of 0.1 m/s (or 0.05 m/s with differential corrections). We use the GPS receiver primarily as a speed sensor. We believe that the accuracy of 0.36 km/h is fully appropriate as we use speed measurements mostly to

provide “context”, while more local changes are contained in the measurements of longitudinal acceleration.

Lassen SK8 is designed to be embedded into third-party products and as such it provides only TTL (Transistor-Transistor Logic) voltage levels for serial ports and does not provide any power management. To simplify the data acquisition system, we decided to use it in the `starter kit’ form – in a metal enclosure with battery backup for fast reacquisition, with added drivers to provide standard RS-232 voltage levels for serial connection, and with a compact active micropatch antenna with magnetic mount. The starter kit also provides sample source code for application development. We used portions of the code from the TSIP sample while developing our data acquisition software, as described in the next section.

4.3.3 The design and implementation of the sensor board

The sensor board was developed to provide for mechanical mounting and electrical connection of inertial navigation sensors. For navigation sensors, the proper orientation is very important. For example, the earth’s gravity influences the measurement of the accelerometer as it moves its common beam out of neutral position unless it is perfectly horizontal. For this reason, precise mechanical mounting was of high importance for board design and implementation. On the other hand, the selected gyroscopes did not require external components. The accelerometers were `embedded’ in evaluation boards, so no additional components were required for them, either. Consequently, the electrical connections required for the sensor board are relatively simple.

In order to measure three-dimensional acceleration and angular turning rate, the necessary sensors (accelerometers and gyroscopes) must be mounted in at least two mutually orthogonal planes. We decided to first implement the printed circuit board for the horizontal plane. It had to mount two accelerometers to measure the vehicle’s longitudinal and lateral acceleration. Also, two gyroscopes could be mounted to measure the rate at which the vehicle turns around its longitudinal and vertical axes. This board, as shown in Figure 4.7, also had connections for another board, which were to be positioned in the vertical plane, with an accelerometer to measure vertical acceleration

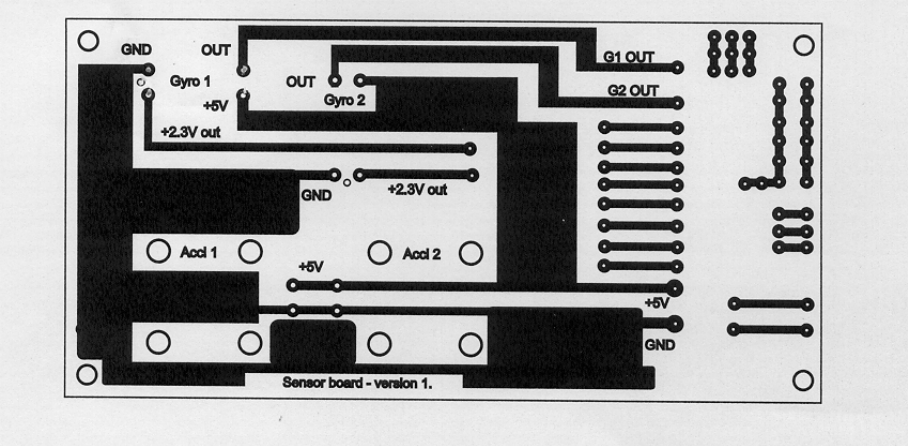

Figure 4.7. The printed circuit design for the sensor board.

and a gyroscope to measure the rate at which the vehicle turns around its lateral axis. The horizontal board was connected to the analogue to digital conversion card.

The positions of each sensor were clearly indicated on the board. The board printwork created electrical connections, and marked mounting holes. Small circles next to the ground (GND) gyroscope connection marked the required holes for the proper mounting of the gyroscopes, which have plastic pins to specify the orientation of the sensor. The four large circles next to each accelerometer marked holes required to mount the accelerator `evaluation' boards.

One of the outstanding problems was to provide a stable power supply for the sensors. Both types of sensor (accelerometers and gyroscopes) require the same supply of +5 V DC. It would have been relatively easy to provide a voltage converter, which would use power from the car's battery as input and generate a stable +5 V output. However, we decided to use the +5V referent voltage source from the A/D card to power the sensors. This source was not designed to power sensors, but the current required by the sensors was low and we did not detect any problems which such a scheme.

We mounted the sensor boards in a simple, box-like aluminium case. We used a metal case to minimise the electrical noise in sensor electronics. Electrical components of a car's motor are known to be a relatively large source of noise. This case made it

easy to install printed circuit boards in two dimensions by drilling the required holes and using insulating spacers.

As we previously stated, the orientation of the sensors should follow the vehicle orientation and should not change during driving, as this may greatly influence measured values. Sensors (with the sensor board and case) should be aligned with the vehicle's longitudinal axis in order to provide proper values for accelerations. Our first option was to drill appropriate holes somewhere on fixed vehicle parts, using bolts to fix the case to the vehicle. This technique would provide proper orientation of the sensors at all times. However, we did not want to modify the car. The other problem with this solution is that the case must be mounted before each driving session, and removed after, as passengers could break the case. We decided to use a much simpler and less elegant approach – we used a large concrete brick and positioned it on the vehicle's floor, aligned with the seat. The sensor board case was attached to the concrete brick with two pieces of rubber, enabling very easy and quick mounting before driving sessions. Due to the brick's large mass, the position of the sensors did not change during driving.

4.3.4 Analogue to Digital Conversion Hardware

An ACL-8112, model DG, data acquisition card was chosen to provide conversion of analogue voltages coming from the accelerometers and gyroscopes into the digital representation required for processing by the computer. The ACL-8112 is a multi-functional data acquisition card that provides high performance and high speed at low cost. It provides the following data acquisition functions: analogue to digital conversion, digital to analogue conversion, timers and counters, and digital input and output. It uses a standard ISA bus for connection with IBM PC compatible computers. The block diagram of the ACL-8112 is shown in figure 4.8.

The ACL-8112 provides 16 single-ended (with a common electrical ground) or eight differential analogue inputs that can be digitised by using a successive approximation method. The input voltage range is software controlled, as the card provides programmable gains of 1, 2, 4 and 8. Input voltages can be unipolar (like 0 – 10 V) or bipolar (±10 V), which can also be selected by the user. The output resolution

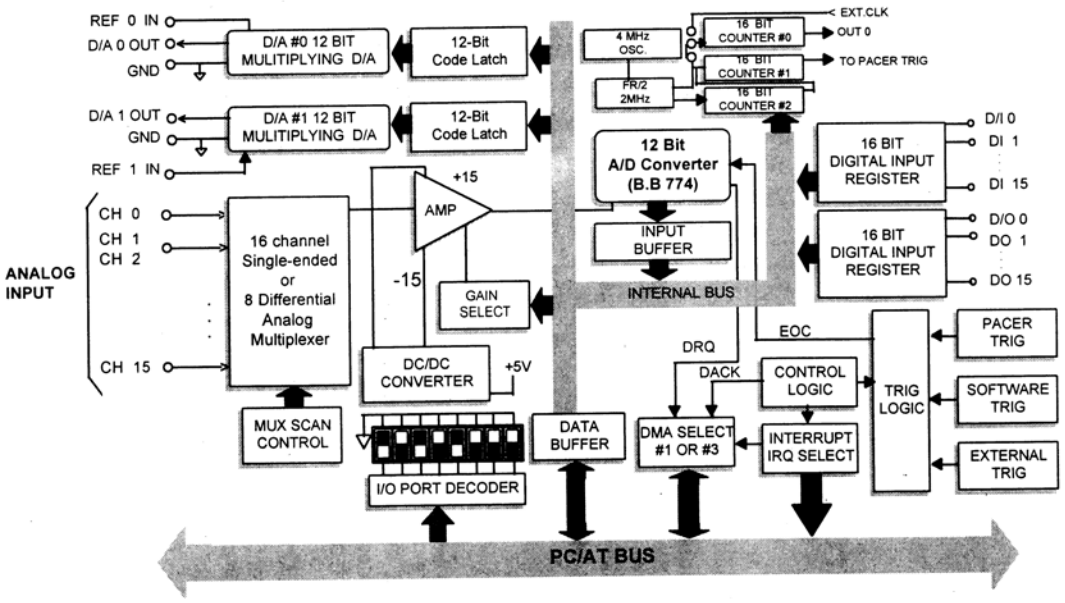

Figure 4.8. ACL-8112 block diagram [ADLink, 1997].

of the converter is 12-bit, which provides resolution of 1.2 mV if the input range 0 – 5 V is selected (the range that we use for reading signals from the accelerometers).

Three different methods can be used to start the analogue to digital conversion process, enabling a wide variety of possible applications to be implemented. These methods are:

- the software trigger from the data acquisition software,
- the internal hardware trigger (pacer) created from counters that are available on the card, and
- the external hardware trigger – a signal that is applied to the dedicated connection.

The rate of the analogue to digital sampling is also programmable up to 100 kHz. The transfer of the digitised data from the card to the computer can be conducted through interrupts, DMA (direct-memory access), or use of the shared input-output (I/O) registers (controlled by the data acquisition software). Interrupts and DMA are common programming techniques used for low-level hardware access in order to enable asynchronous operation (interrupts) or faster data transfer speed (DMA). In both cases,

low-level programming is required (effectively writing a device driver for the card). Our requirements were easily fulfilled by using I/O registers, so we decided to use software triggers and software control over the data transfer process.

The ACL-8112 contains two double-buffered analogue outputs. They accept 12-bit digital information input and output analogue signal in the range 0 – 5 V (or 0 – 10 V, selectable). There are also 16 digital inputs and outputs for interfacing with TTL circuits. In addition, the card contains three independent 16-bit down counters that can be inter-connected for a number of different counting and timing applications. An internal 2 MHz oscillator or external clock source can be used to trigger counters. The outputs of the counters can be used to trigger analogue to digital conversion (the hardware pacer method) or for other application-specific uses.

The card uses an ISA bus for connection with a PC computer. ISA is a 16-bit bus that was introduced with IBM AT computers, and is also known as a PC AT bus. An ISA bus has a 98 pin edge connector that can still be found in some older motherboards on the market. The main communication medium between the card and the computer is a small amount of the shared memory in the I/O memory space of the computer. ACL-8112 has sixteen 8-bit memory locations (registers) in a shared I/O memory space. Through these the computer sets values and controls the operation of the card. The card uses the same memory locations to store the state of the A/D converters, counters or digital inputs, and the computer can read them, if required. The base address for registers is switch selectable but constant, which makes the card programming much easier than that of modern PCI cards, and its use does not require special device drivers

4.4. Testing accelerometers and the AD card

After the sensors, mobile computer, and GPS receiver were integrated, we performed a number of tests in order to explore the usability of the system. We firstly conducted a test with stationary sensors in order to deter if they were working. To test the GPS receiver, we used the sample application (TSIPCHAT) provided with the Lassen SK-8 development kit. The output from the receiver is shown on the computer monitor immediately after the receiver is connected to the power and the computer. The data acquisition card had a similar utility that displayed expected voltage levels for connected

signals. However, this utility did not provide the option to save collected values for future evaluation. To overcome this problem we developed a very simple program to test the data acquisition card. This program reads a large number of digitised values from one channel of the data acquisition card and stores them in memory. Once reading is finished, the program saves digitised values in a file for future processing. The most important part of this program's source code is presented in Figure 4.9.

The immediate results of tests with the developed utility are presented in the diagram in Figure 4.10. and a small part of the data is presented in its numerical form in Table 4.1. From these data we were able to calculate that the data acquisition rate was approximately 50 kHz, because the time difference between the two collected values

```
// high precision timing
__int64 start, end, freq;
if(!QueryPerformanceFrequency((LARGE_INTEGER*)&freq))
{
 AfxMessageBox("No time support");
 return;
}
double  dur = 1.0/freq;
// setup card
unsigned int m_base = 0x300;
_outp(m_base + 11, 0x01);  // setup sw polling mode
_outp(m_base + 10, 0x30);  // select channel 0
_outp(m_base + 9, 5);      // set gain 5 = 0-5V
// collect data
int i = 0;
QueryPerformanceCounter((LARGE_INTEGER*)&start);
while(i < BuffSize)
{
  _outp(m_base + 12, 0x55); // sw trigger - start conversion
  // read data until DRDY is high
  unsigned char ByteHi, ByteLo;
  do
  {
    ByteHi = _inp(m_base + 5);
  }
  while(ByteHi & 0x10);

  // concatenate bytes
  ByteLo = _inp(m_base +4);
  unsigned short Data = ByteHi & 0x0f;
  Data = Data << 8;
  Data = Data | ByteLo;
  ValueAr[i]   = Data;
  // get time stamp
  QueryPerformanceCounter((LARGE_INTEGER*)&end);
  TimeAr [i++] = (end – start) * dur;
}
```

Figure 4.9. The source code for the program to test the analogue to digital conversion card. It reads digitised values from the card and stores them to the memory array ValueAr. The second array (TimeAr) is filled with the precise time of data collection.

Units	Time (s)
1956	0.008529
1957	0.008549
1928	0.008568
1924	0.008587
1949	0.008606
1924	0.008625
1914	0.008645
1956	0.008664
1952	0.008683
1900	0.008702

Table 4.1. A short sample of the collected data from acquisition test program.

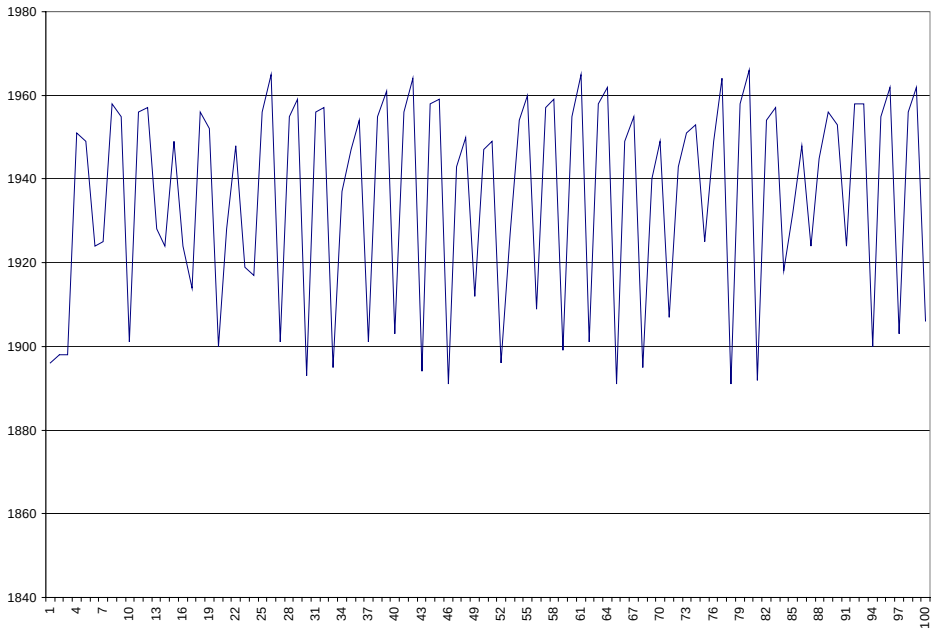

Figure 4.10. Graphical representation of a part of the collected data.

was approximately 0.000020 seconds. This rate was more than appropriate for our application.

The collected digitised values are in the range of 1900-1960 units. The input voltage range was set to 0 - 5 V DC and the resolution of the conversion was 12-bit, so calculated input voltages were 2.32 V and 2.39 V respectively. The mean value for input voltage was 2.37 V and the standard deviation was 0.028 V.

Another small program that we developed used a Fourier transform to convert collected data from the time domain into the frequency domain. From the generated data, we concluded that the main noise source for accelerometers had a frequency of approximately 15 kHz. The frequency for gyroscope noise source was approximately 8 kHz, which is consistent with ENC-05D documentation, which declares that the prism in the sensor oscillates with a frequency of 8 kHz. In both cases, noise frequencies were much higher than frequencies we were interested in, as vehicle events are relatively slow and usually last a few seconds. We concluded that we would not need additional hardware or sophisticated software to remove noise or improve the signal to noise ratio.

Since sensor behaviour was within expected limits, we now began the next set of experiments. The sensors and the mobile computer were still on the office desk, but we manually moved sensors to check their responses. We tried to limit motion or rotation of sensors to one direction or axis (depending on which sensor we tried to test). Tested sensors were sensitive enough to enable visual detection of fast manual motion in the collected data displayed in a graph.

The final test was conducted with sensors and mobile computer fitted inside the car; several test drives were conducted with this setup. The software was designed and implemented to work in the same fashion as embedded systems without user interfaces: it started as soon as the computer was switched on and the operating system was loaded. The system quietly collected data, and finished when the computer was switched off. The collected data were saved in a file. We soon experienced problems with this design – there was no way to determine visually whether or not the system was collecting data, so a few long test drives left us without any data. If we wanted to collect data on two different routes and separate them into different files, we had to restart the computer. Long sequences of data were also hard to analyse visually, which was our initial data analysis preference. Only when we returned the mobile computer to the office and retrieved the collected data, were we able to comprehend the results of our driving tests. The obvious conclusion was that some sort of user interface was necessary to avoid such problems in future. The other important conclusion was that the necessity to set up the mobile computer in the office only to retrieve data should be avoided if possible.

Results collected from accelerometers in this system were good, as it was possible to visually identify all important driving events (starts and stops, turns, and so on). The

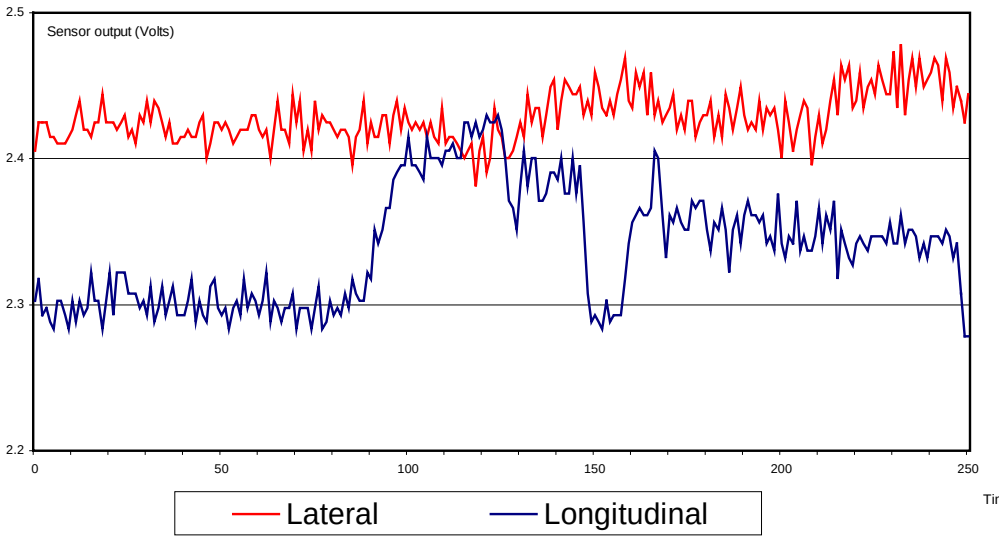

Figure 4.11. The `raw' data collected from accelerometers during driving tests.

signal noise was evident; however, basic waveforms were easy to identify, as shown in Figure 4.11. As we noted previously, during these tests we found that the sensitivity of gyroscopes was not appropriate for automotive applications. We were unable to distinguish sensor output from noise at any time; thus we decided not to use gyroscopes in further experiments and rely on data from accelerometers instead.

4.5. Data acquisition system hardware

The designed data acquisition system and selected sensors generally performed well in the tests described in the previous chapter. In this section, we describe the final architecture of the data acquisition hardware and how it was used for data collection.

Test drives clearly indicated the need for some form of user interface to indicate that the system was running, as well as the state of some components. A data input hardware component was also necessary to provide the ability to mark sections of the driving data that had some particular meaning. Finally, we also needed a method for "automatic" data transfer to the office computer.

In order to fulfil these requirements, we decided to expand our mobile computer with a keyboard, sound card, and speakers. For data transfer, we decided to use a floppy drive controlled by the data acquisition software.

Our data input requirements were minimal and easily solved by a very simple keyboard with only a few keys. However, because of their infrequent use, such keyboards are much more expensive than a standard, complete PC keyboard. To minimise cost, we decided to use a standard keyboard, which was permanently fixed to the top side of the mobile computer with double sided tape. Because we needed only a few commands, we chose four function keys (distant from each other) on the top of the keyboard, clearly marked them, and designed our data collection software to recognise when they were pressed. The data acquisition software ignored all other keys on the keyboard.

Recent research in vehicle navigation [Petkovic, 1997] demonstrated that drivers' visual communication channels are already filled with a huge amount of visual information about the environment and that inclusion of additional visual sources in the drivers' `cockpit' might be a major source of traffic accidents. For this reason, many modern vehicle navigation systems include voice-based communication channels. We decided to follow the same trend and we equipped our mobile computer with the simplest audio card and speakers. Data acquisition software was designed to provide audio feedback, which will be explained in more detail later.

The final architecture of the data acquisition hardware is presented in Figure 4.12. All components, connected as in use, are shown in Figure 4.13. The mobile computer was the central component to which speakers, sensor board, and the GPS receiver (with an antenna) were connected. When connected to the car's cigarette lighter the power inverter provided required 220 V AC power for the computer. Because the cigarette lighter was already in use, the GPS receiver was powered through a separate power supply. It is clear from the picture that the only custom-made component of our system was the sensor board; all other components were off-the-shelf products that allowed the very low price and reliable operation of the system. Figure 4.14. illustrates how the data acquisition hardware components were installed into the vehicle. The mobile computer was located on the front passenger seat, and the seat belt was used to fix it in this

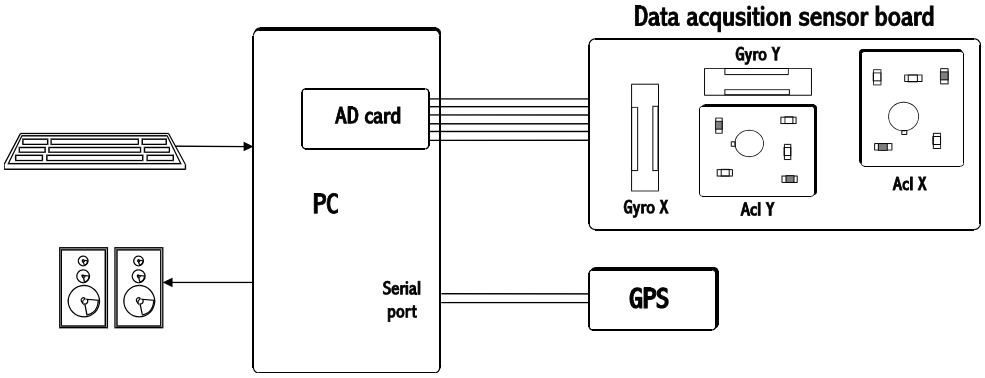

Figure 4.12. The final architecture of the data acquisition hardware.

Figure 4.13. All components for the data acquisition hardware.

Figure 4.14. The data acquisition hardware installed in the car and ready for use.

position. The power supply was located on the car floor in front of the computer. The GPS receiver was located on the dashboard, and the GPS antenna was simply magnetically mounted on the vehicle's roof. Speakers were located on a back seat, while the sensor board (fixed in its metal box) was tied to a brick on the car floor below the passenger seat (not visible in the image). As can be seen, no changes to the vehicle were made, so standard vehicle configuration was preserved.

4.6. The design and implementation of the data acquisition software

The purpose of the data acquisition software was to:

- provide an accurate and robust data collection tool,
- to enable easy data transfer from the mobile to the office computer,
- to convert data to the format(s) required for experiments in learning driving patterns,
- to enable a limited, but functional user interface,
- to be easy to maintain, and
- to be extensible with real-time learning modules if/when required.

The data collection process was the combination of two mostly independent tasks: collection of data from accelerometers through the A/D card, and collection of velocity and position information from the GPS receiver, as shown in Figure 4.15.

We decided to collect data from accelerometers with a frequency of 20Hz (at 50 ms intervals). This frequency was much higher than required for the vehicle motion tracking, but still much lower than the maximum frequency the data acquisition

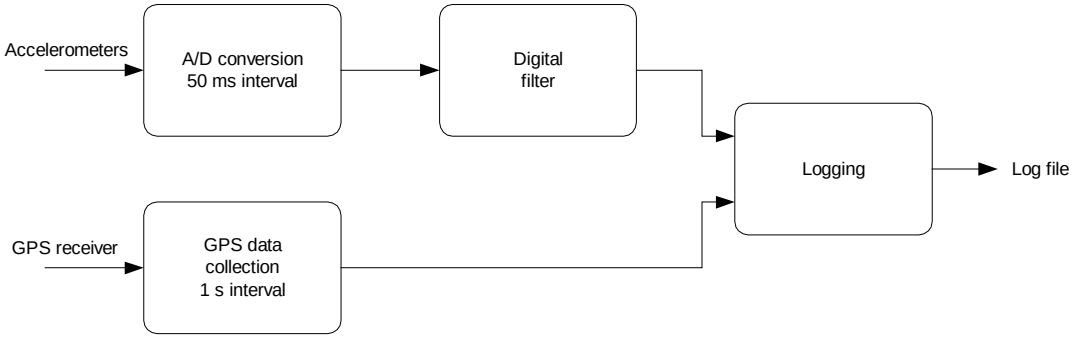

Figure 4.15. The data collection process.

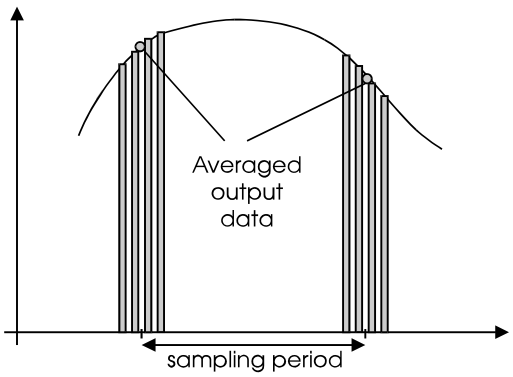

Figure 4.16. The averaging of collected data.

hardware and the processor in the mobile computer could handle. We implemented provisions in the data acquisition software to configure collection intervals to any number of milliseconds through registry setting, but we never experienced the need to do so.

A simple averaging technique was used to increase the signal-to-noise ratio, working as a simple low-pass filter. The data acquisition software waited for 50 ms after the start of the last data collection; then it asked the A/D card for 5 independent and consecutive measurements for each accelerometer. The total duration of these 5 measurements was only 0.1 ms, which was very short compared to the data collection interval. These measurements were averaged into one, and that figure represented the final digital value of the measured acceleration, as displayed in Figure 4.16.

Due to the mechanical nature of the vehicle, changes in acceleration and speed were very slow, and we were not interested in frequencies higher than 5 Hz. Higher frequencies represent noise that could cause problems with our later experiments, so we decided to install a proper low-pass filter to eliminate this noise. We decided to use a second order normalised low pass Butterworth digital filter with a cut-off frequency of 2 Hz. Butterworth filters have monotonic magnitude response, as shown in Figure 4.17. After applying this filter, the signal on lower frequencies will preserve its original magnitude. The relatively wide pass-band was not a major problem, because noise levels are low for low frequencies.

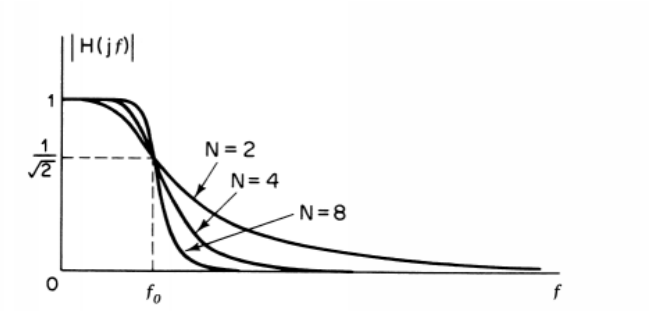

Figure 4.17. Butterworth filter response in the frequency domain.

The transfer function of the second order low-pass Butterworth filter is given as: $H(s) = \frac{1}{s^2 + \sqrt{2}s + 1}$ [Stearns, 1988]. The filter is designed by using substitution $s \leftarrow \frac{s}{\Omega_c}$ where $\Omega_c = \tan(\frac{2\pi f_c T}{2})$. Here, f_c is the selected cut-off frequency (2 Hz) and T is the data acquisition interval (50 ms). Mapping from the analogue to digital domain is accomplished using a standard bilinear transformation $s = \frac{z-1}{z+1}$. The resulting digital transfer function can be expressed as $H(z) = \frac{b_0 + b_1 z + b_2 z^2}{1 + a_1 z + a_2 z^2}$. Figure 4.18 shows the second order, low-pass Butterworth filter with cut-off frequency of 2 Hz implemented as an MS Excel macro.

An important requirement for implementing a simple digital filter such as the one just described is to provide normalised data with a mean value equal to zero. To provide such data, we designed a simple normalisation method to provide continuous normalisation (over all data collection sessions). The data collection software keeps minimal, maximal, and mean values for each sensor persistent in the Windows system registry. These values are updated after each new data item is collected so that final data was normalised over all previously collected data. Immediately before digital filtering, each value was normalised by the respective mean value, and after filtering, the values were de-normalised using the same value; thus, the original range is preserved.

```
Sub ButterworthFilter(Rows As Integer, InputData As Range)
‘ initialisation
    PX1 = 0
    PX2 = 0
    PY1 = 0
    PY2 = 0
    ‘ filtering
    For k = 1 To Rows
      PX0 = InputData.Cells(k, 4)
      Sum = 0.0675 * PX0 + 0.1349 * PX1 + 0.0675 + PX2
      Sum2 = -1.1429 * PY1 + 0.4128 * PY2
      Sum = Sum - Sum2
      PX2 = PX1
      PX1 = PX0
      PY2 = PY1
      PY1 = Sum
      InputData.Cells(k, 5) = Sum
    Next k
End Sub
```

Figure 4.18. An MS Excel macro implementing
the second order low-pass Butterworth filter.

Before we selected the cut-off frequency, we tested several possible values. Figure 4.19 shows normalised but unfiltered values for longitudinal acceleration, as well as filtered values for cut-off frequencies of 1, 2, and 5 Hz. These data represent the vehicle

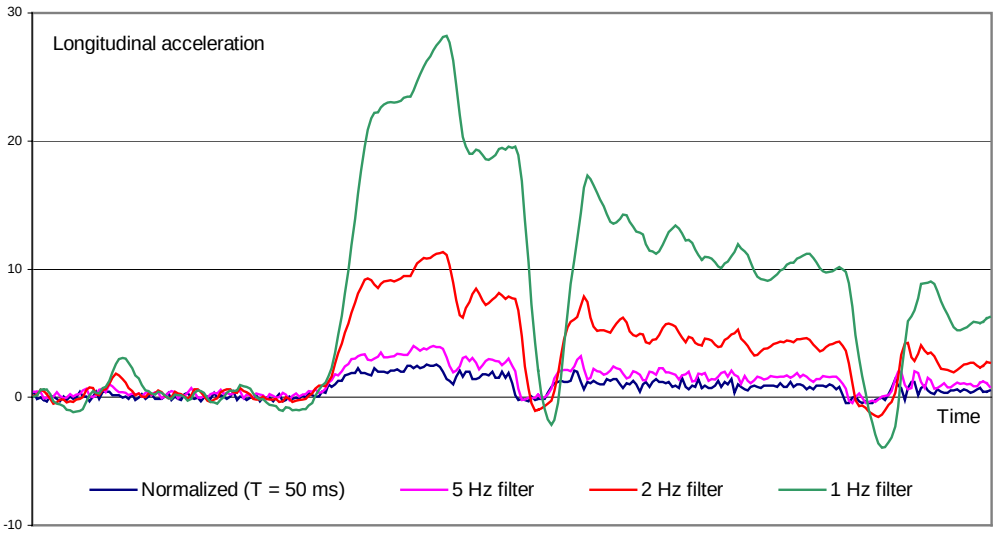

Figure 4.19. Normalised and filtered values for longitudinal acceleration
for various cut-off frequencies.

starting, the change from first to second gear, and then the shift from second to third gear. The increase in signal-to-noise ratio is well demonstrated, because filters with lower cut-off frequencies produce values in which waveforms are much easier to visually recognise.

The drawback of filtering is the delay introduced into the filtered signal, and this delay gets longer the lower the cut-off frequency. We selected the cut-off frequency of 2 Hz as it represents a convenient compromise: the signal is clearly distinguished from noise, and the delay is small compared to the duration of elementary driving actions such as a gear change. The advantages of the 2 Hz filter are clearly visible in Figure 4.20.

The GPS receiver communicates with computers through a serial port using the Trimble Standard Interface Protocol (TSIP). TSIP is based on the transmission of

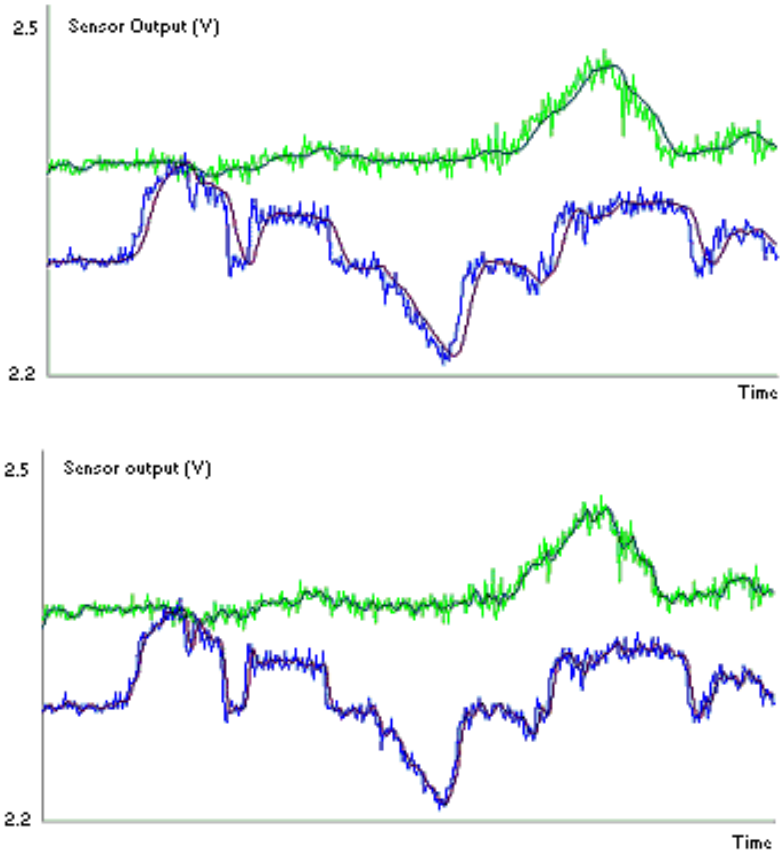

Figure 4.20. Original and filtered longitudinal and lateral accelerations for the 1 Hz filter (above) and the 2 Hz filter (below).

packets of information. There are over seventy TSIP commands that can be used to configure a GPS receiver for optimum performance and the selected application. There are additional reports through which the GPS receiver sends data about current position and velocity, the state of the receiver and the GPS system, as well as other data. Some of these reports are generated in response to commands, but the receiver is set to automatically send some reports at predefined intervals without any prompt by the computer.

We were interested in collecting position and velocity information from TSIP records. We also extracted the status information to enable system troubleshooting – to inform us while the system performed its initialisation (which could be long, particularly if no data collection had been conducted recently), if there were not enough visible satellites, or if some other problem caused the position and velocity information to be unavailable.

After the receiver successfully performs initialisation and starts tracking enough satellites, the position and velocity records are reported at 1 second intervals. This was slower than the 50 second intervals used to collect data from the accelerometers. Receivers that can provide position reports with regular frequency are available, but are significantly more expensive than the one we selected. Fortunately, the vehicle changes in position and velocity were relatively slow, and the interpolation of data between the two reports provided good results, as demonstrated in Figure 4.21.

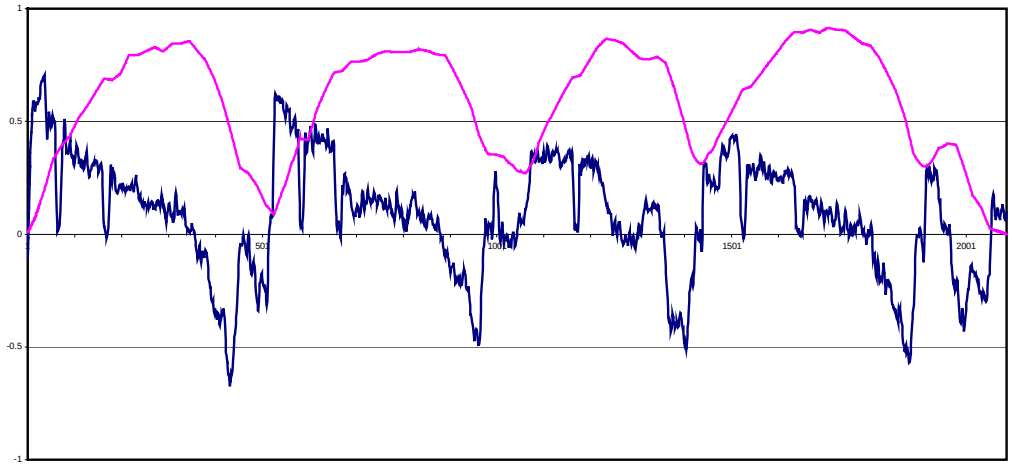

Figure 4.21. Normalized longitudinal acceleration and vehicle's speed.

The data acquisition software was designed and implemented using object-oriented techniques and tools. We used a simplified version of the Coad-Yourdan method [Coad, 1991], an object-oriented design and analysis methodology that we had used for almost a decade in everyday professional software development. We found the data acquisition software to be a comparatively small system and easy to design. As shown in Figure 4.22, there are only 7 classes that implement main functionalities of the system. For programming, we used Microsoft Visual C++ versions 5 and 6 and a Visual Studio development environment.

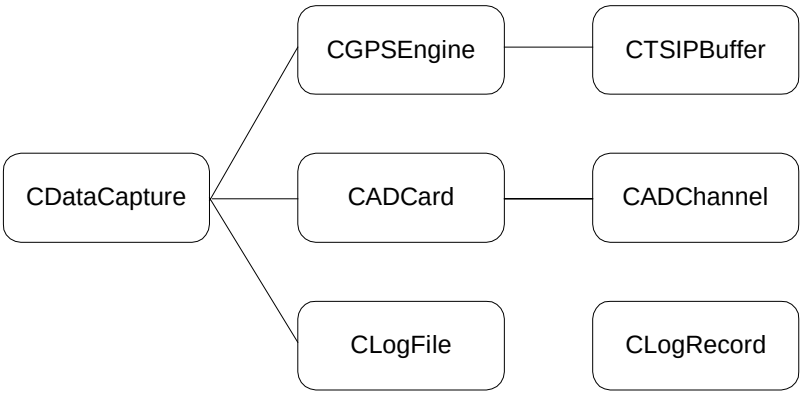

Figure 4.22. Classes used for implementation of the data acquisition software.

The *CGPSEngine* class encapsulates all aspects of work with the GPS receiver. When higher level classes require the *CGPSEngine* to start, it creates a new thread, sends initialisation commands to the receiver, and listens to the serial port for data from the receiver. *CTSIPBuffer* encapsulates the TSIP protocol so that it does not have to be contained in the *CGPSEngine*. It encodes commands into the TSIP protocol and sends them to the receiver. Data received from the GPS receiver are split into separate packets by searching for the "<ETX>" byte (0x03), which marks the end of the packet. Packets are further unpacked from their binary form into a form compatible with a C++ interface. The data received for selected packets (position, speed, time, and the receiver status) are passed to the *CGPSEngine* class, which passes them to the higher-level classes using the callback (also known as subscribe-publishing) technique.

The class *CADCard* encapsulates all aspects of the actual A/D card relevant to the acquisition of data from analogue sensors. To further simplify the design, we developed the class *CADChannel*, which deals with one particular sensor. At the start of the data

collection session, the *CADCard* initialises the actual card, and analogue-to-digital conversion parameters (input voltage range) for each required sensor/channel are set. To simplify the overall design, *CADCard* also operates in its own thread. It waits for a specified amount of time (50 ms) and then collects data from all requested sensors. The collected data are propagated to the instance of the *CDataCapture* class.

The *CDataCapture* class controls the operation of both the GPS receiver and the A/D card. When a data collection session is requested, it initialises both classes and creates an instance of the *CLogFile* class, which creates a new file on the disk. The file name is built from the current date and time so that the system can operate without human intervention and allows several data collection sessions during a single run. The instance of the *CDataCapture* class receives collected data from the receiver and sensors and passes them to the *CLogFile* by using the *CLogRecord* class. The *CLogRecord* class has data members to store collected measurements from sensors, speed from the GPS receiver, and the time of the collection. Most instances of the *CLogRecord* class do not have speed data, because speed is logged only at 1 second intervals. The *CLogFile* class provides a buffer for all *CLogRecord* instances that do not have speed data. When the *CLogFile* receives an instance of the *CLogRecord* class with a proper value for the speed, it interpolates speed values for all records in the buffer, and then writes all records to the disk file.

When the driver decides to close a data collection session, the instance of the *CDataCapture* class deconstructs all instances of other classes, which causes data collection to stop, remaining records to be written to the log file, and the session to be closed. If the driver decides to end operation of the data acquisition software, the *CDataCapture* packs all log files found in the folder designated for collected data into a zip file and will writes it to the floppy disk. Log files are then moved to the 'archive' folder.

To implement the system's audio feedback we used a very simple technique. We identified the most important states and possible error conditions where data acquisition software will typically display a warning or an error message. For each such message we made an audio recording in a ".wav" file, using the standard Window's Sound Recorder tool. We minimized the size of the files by reducing audio quality, which is not important in noisy vehicle conditions. These ".wav" files are integrated in the software

as resources alongside menus, icons and other elements of the user interface. At each place where we needed to present a message in the software, we called the Windows API (Application Programming Interface) method *PlaySound*. This method plays recorded sounds via the connected speakers.

To prove that pattern learning techniques could be integrated into an existing vehicle navigation system without much additional work, we extended our data acquisition software with some functions usually found in vehicle navigation systems. The extended system is able to display the vehicle position over the raster map of the area (in the New Zealand Map Grid coordinate system). As shown in Figure 4.23, the current position is marked as a large cross in a light green colour, and the last 15 seconds of the vehicle's path are represented as a 'tail' in a similar colour. The system also has moving map capabilities – when the vehicle position moves to the window border, the map is 'panned' and the cross marking the vehicle position is set to the centre of the window. The system is also able to save and retrieve routes travelled and significant points (waypoints) marked by the user. Our mobile computer did not have

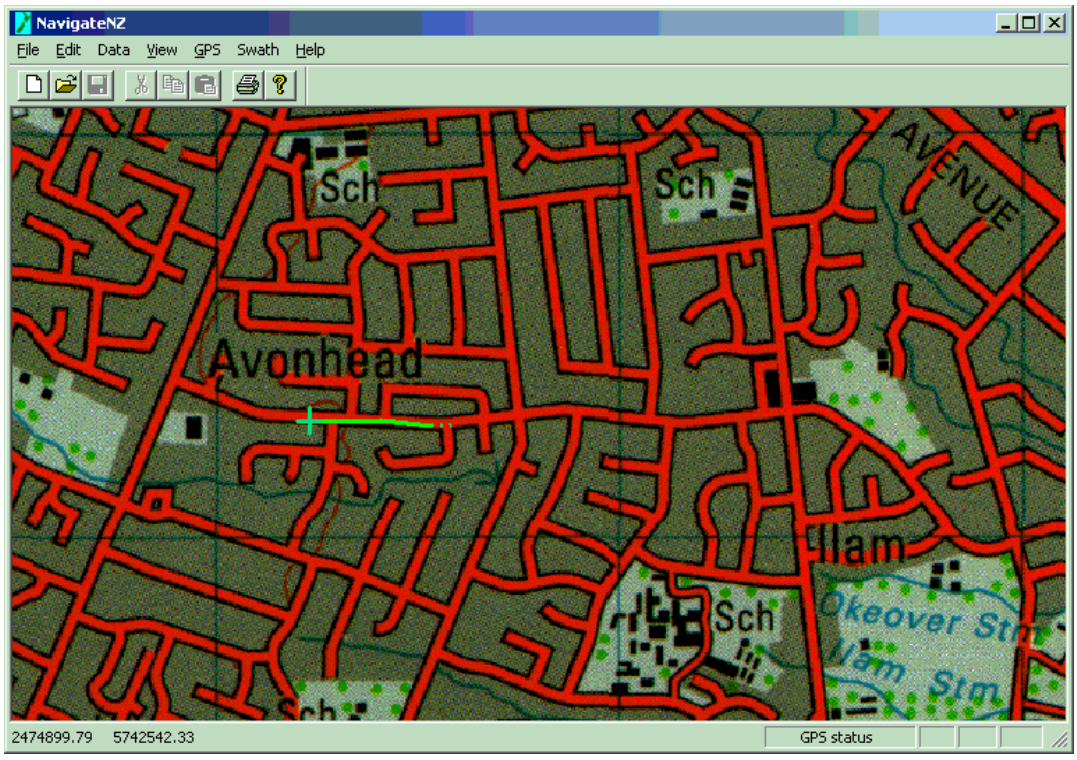

Figure 4.23. The output from the extended data acquisition system.

any display capabilities (in mobile mode), so we borrowed a laptop to test these functions.

Before we could use the collected data as part of pattern learning experiments, several transformations of the data must be performed: data must be read from the binary log file, visualised to confirm the validity of the collected data, mapped into predefined input ranges for experiments, and saved in a format required by the software used in our experiments. Since spreadsheets are probably the most powerful and versatile general-purpose, 'off-the-shelf' programs we decided to use Microsoft Excel. Excel provides easy manipulation of data in any form, as well as visualisation and storage to a variety of formats. Additionally, Microsoft Excel has a fully functional programming language (Visual Basic for Applications, VBA) for developing additional functionality when required.

We could have developed an Excel macro to read collected data from log files. However, due to differences between C++ and VBA data models, we concluded that it would be easier and safer to implement a small utility program in C++, which reads the log file and saves data as a text file. The text file is then easy to load into Excel. After the collected data are loaded in Excel, it takes only a few keystrokes to generate a chart. A simple map of collected positions (in the New Zealand Map Grid coordinate system) created in Excel is shown in Figure 4.24. Many other figures of collected and predicted

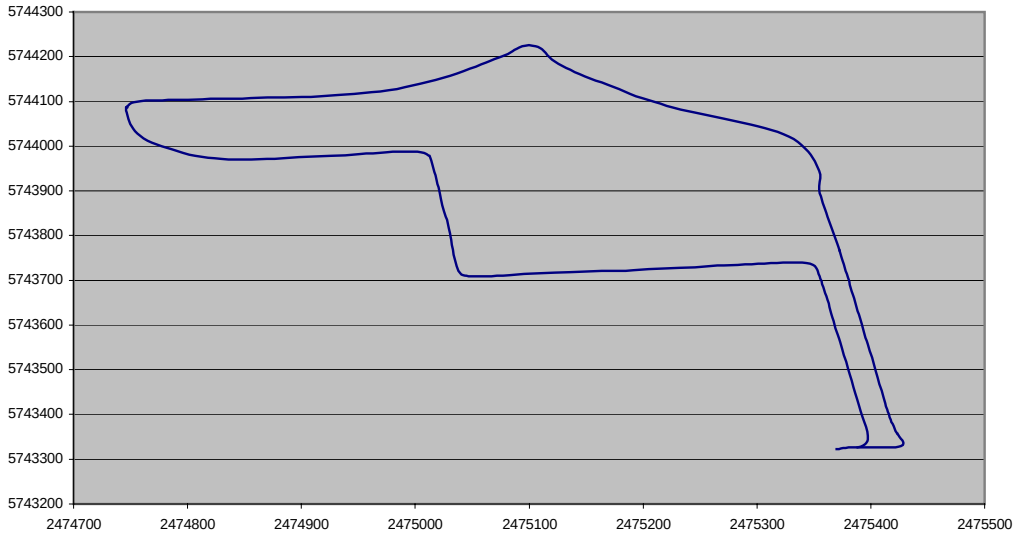

Figure 4.24. A simple map created as an Excel chart from collected positions.

data in this and other chapters were generated in Excel. We also developed simple macros to transform data from original values into the ranges required for our experiments. This data was extracted and saved in whatever format was required for any particular purpose.

4.7. Conclusion

In this Chapter we described the complete process of designing and building the data acquisition system. We needed the data acquisition system to collect data that characterises a person's driving from sensors and to make it available for experiments with learning of driving patterns.

The main factor influencing the design of our data acquisition system was a very limited research budget. Another important limitation we faced was the use of a private car, which we wanted to maintain in original condition. Due to these limitations we had to make some unorthodox choices in the system's design.

As a result we built a mobile computer from the components typically used in desktop computers. It was powered by the car's battery using a power inverter. We used a multifunction analogue-to-digital conversion card to collect data from sensors. To provide a limited user-interface our mobile computer was equipped with keyboard and speakers. Our system consisted mostly of 'off-the-shelf' components, which reduced cost and increased flexibility and extensibility of the system.

We selected the micro-machined accelerometers ADXL05 to measure the vehicle's longitudinal and lateral acceleration. We planned to use micro-machined gyroscopes to measure turning rates; however, the gyroscopes that were within our price range did not have appropriate sensitivity. We found that the vehicle's turning rate could be estimated from the lateral acceleration data, which clearly indicates any vehicle turn.

The design and implementation of the data acquisition software followed a well-known object-oriented methodology. Only seven classes were required to build the core data-collection part of the system. The functionality of the classes is clearly and logically separated and their internals encapsulated, making the code easy to implement and maintain. We also easily extended the data acquisition software by adding classes

for displaying the current position of the vehicle over a geo-referenced background map. The moving map capability, which is common for car navigation systems, was also added.

The data acquisition system was successfully used to collect a large amount of data for our experiments with driving pattern learning. This system was later also used by Kerryn Offord, a postgraduate student in the Psychology Department, University of Canterbury, without any modification,.

5. Short Term Prediction of Vehicle Motion by Neural Networks

Goals:

The goal of this chapter is to present the results of experiments in short-term vehicle motion prediction using neural networks. Neural networks are computational models developed to mimic human information processing. Short-term prediction has quickly become a very popular application of neural networks, as they are well known for their prediction capabilities. We conducted experiments to explore the possibilities of short-term prediction of vehicle movement by neural networks and to compare prediction performances of various neural network architectures.

Basic neural network concepts are briefly outlined, followed by some examples of applications of neural networks for prediction. Next, a short survey of neural network applications in intelligent vehicles and navigation is presented. Our experiments in short-term prediction by neural networks are described in detail and the results are discussed.

Results described in this chapter have been published in the following papers:

Mitrovic D., Experiments in Subsymbolic Driving Pattern Prediction, in *Proceedings of the International Conference on Neural Information Processing*, pp. 673-678, 1999.

Mitrovic D., Short term Prediction of Vehicle Movements by Neural Networks**,** in *Proceedings of the International Conference on Knowledge-Based Intelligent Information Systems*, pp. 187-190, 1999.

5.1. Neural networks

The brain is the most fascinating, but also the most complex and unexplored part of the human body. The brain can be regarded as a complex, non–linear and parallel information processing system. The structure of the human brain was largely unknown until the twentieth century. Even now, many aspects of its functioning remain mysterious.

Nervous systems in different animals possess global architectures of variable complexity. However, all are built from similar building blocks: neural cells or *neurons*. This idea of neurons as the basic building block of the brain was introduced by Ramon y Cajal in 1911 [Cajal, 1911]. There are many different types of neurons, but they all have four basic components: dendrites, synapses, cell body (soma) and axon, as depicted in Figure 5.1.

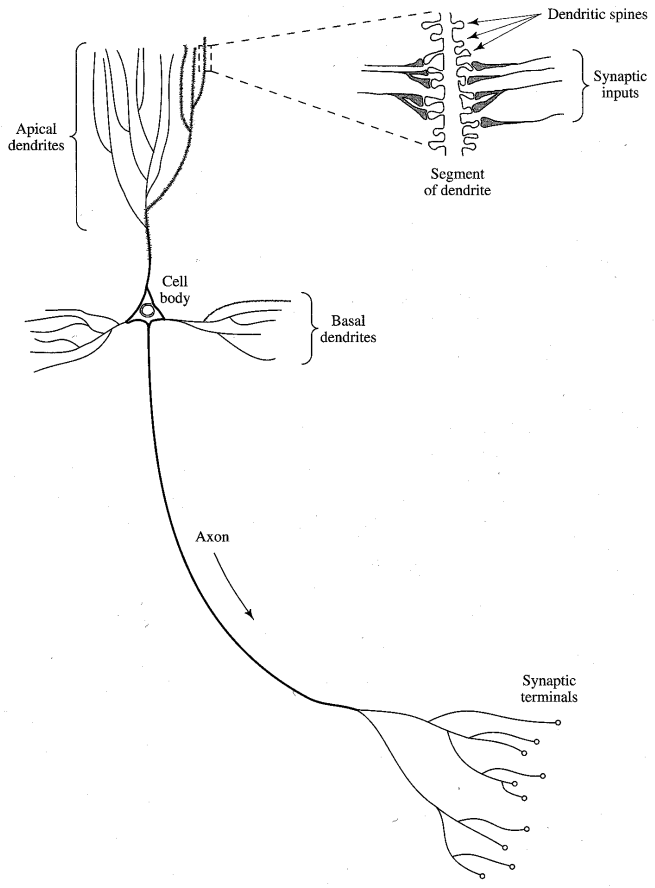

Figure 5.1. The structure of the pyramidal neural cell, from [Haykin, 1994].

Dendrites are transmission channels for incoming information. They have contact regions where other cells are connected known as *synapses*. Electrical and/or chemical transmitters carry out information transfer at the synapses. Generally, an incoming electrical signal from an axon is used to generate chemical transmitters in the synaptic terminals, which are released into the synaptic gap. These chemical transmitters generate a new potential difference on the synapse's wall of the receiving neuron. The strength of the electrical signal transmitted to the dendrite is influenced by a number of factors. A signal received by a dendrite may result in its excitation, a positive contribution to the dendrite's output, or inhibition, a negative contribution. The dendrites transport the electrical signal to the *soma*. If the sum of the input signals from the dendrites is larger than the soma activation threshold, an activation potential is generated at the *axon*. The generated signal known as *activation potential*, is transmitted through an axon to excite other neurons.

The activation potential and all the other electrical operations in the neuron are conducted by opening and closing ionic channels and pumps in the cell membrane. Different concentrations of potassium (K+) and sodium (Na+) ions on the cell's membrane generate an electrical potential. While cells rest, the potential is -70mv. However, the activation potential can reach +40 mV when fully excited.

Information processing in neurons is conducted in synapses, where the input signal is amplified by some amount, and in soma, where incoming signals are summarized and compared against the activation threshold. Information is transmitted by activation potential spikes - the frequency of the activation potential is the main source of information.

Short-term memory in a neuron is usually associated with spatial arrangements of membrane potential levels (activations) of many neurons which result from recent events. Long-term memory is associated with amplification factors in synapses and the value for the activation threshold in the soma. These store the results of neuron experiences over a longer period.

At this point we should note a striking difference between brains and today's digital computers. The brain is much slower (10^{-3} seconds per operation compared with 10^{-9} seconds per operation in modern CPUs), but is massively parallel (10^{10} neurons), highly interconnected ($6*10^{13}$ synapses or connections), and much more redundant and

energy efficient (e.g. 10^{10} times less consumption of power per operation per second than modern CPUs). Another interesting point related to brain architecture is that there are only about 100,000 genes in the human genome. Comparing this with huge complexity of the human brain we can conclude that the circuitry of the brain is epigenetic; i.e. determined to a large extent by the environment.

5.1.1 Computational models of the neuron

Advances in understanding neuron function coincided with interest in models of computation conducted by Turing, von Neumann, Hilbert, Kleene, and others. As a consequence, early attempts were made to devise a computational model which would mimic information processing in the human brain. In 1943 McCulloch and Pitts published the first model of a neuron [McCulloch, 1943]. Their model has a number of excitatory inputs (denoted as x_i) and a number of inhibitory inputs (denoted as z_i) as shown in Figure 5.2. If there are no inhibitory signals, the total excitation is computed as the sum of the signals on the excitatory inputs and compared with the threshold value. If the sum is larger than the threshold value, the neuron fires 1 as output. If there is at least one inhibitory signal, or the sum of the excitatory signals is less than the threshold, the neuron's output is 0. Mathematically, the neuron's output is calculated using the following formula:

$$y = \begin{cases} 0, & z_i = 0, \forall i, i = 1, m \quad or \quad \sum_{j=1}^{n} x_j < \theta \\ 1, & \sum_{j=1}^{n} x_j > \theta \end{cases}$$

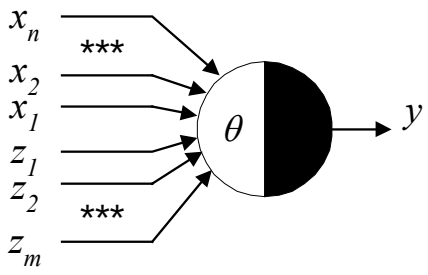

Figure 5.2. McCulloch and Pitts' model of a neuron.

McCulloch-Pitts' model of the neuron is limited by the fact that only binary values are possible as outputs. However, it contains all the necessary features to implement more complex models. McCulloch-Pitts' neurons can be used to build networks which can compute any logical function and can simulate any finite automata. Another significant limitation of McCulloch–Pitts' model is the lack of free parameters, which means learning is possible only by changing the network topology.

In 1958, a more general computational model called the *Perceptron* was proposed by Rosenblatt [Rosenblatt, 1958]. The perceptron was, in fact, a whole network for the solution of certain pattern recognition problems. Minsky and Papert [Minsky, 1969] analyzed perceptrons and distilled an elementary computing unit – which is usually referred to by the same name – perceptron. Formally, the only difference between perceptron neuron and McCulloch-Pitts' neuron is the presence of weights in the input connections. The perceptron model of a neuron is shown in Figure 5.3. It consists of two parts: a linear combiner and a hard limiter. If a neuron has n input signals $x_1, x_2, \ldots, x_n$ and the corresponding weights are $w_1, w_2, \ldots, w_n$, the linear combiner calculates the difference between the weighted sum of the input values and the threshold $\sum_{i=1}^{n} w_i x_i - \theta$.

The hard limiter translates the output from the linear combiner into output values appropriate for the model. The perceptron has two possible output values –1 and +1. The function φ is called the *activation function* and, in case of a perceptron, this is a step (threshold) function as defined by the formula:

$$y = \begin{cases} +1, & \sum_{i=1}^{n} x_i w_i - \theta \geq 0 \\ -1, & \sum_{i=1}^{n} x_i w_i - \theta < 0 \end{cases}$$

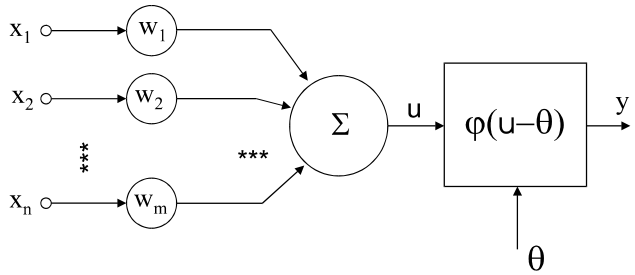

Figure 5.3 Perceptron model of a neuron.

It has been shown that decision characteristics of the perceptron are not related to the choice of the activation function, and the hard limiter is as equally useful as other more complex functions described later.

In the beginning, Rosenblatt's model of the neuron aroused unrealistic expectations in the computational research community. It seemed as though neural networks could do anything a Turing machine could do. However, Minsky and Papert demonstrated that there are fundamental limits to what a *one-layer* perceptron can compute – linearly separable functions. This means that a perceptron cannot compute a simple XOR function, for example. This fact, together with the very limited computational power available at the time inhibited progress in neural network research during the next decade. However, the introduction of input weights was an important step, as it provided a more flexible model and paved the way for research in neural network learning techniques.

The next step in neural network development was the ***Adaline*** (Adaptive linear element) model. Adaline was originally developed by Widrow and Hoff [Widrow, 1960] as an adaptive pattern classification machine. The major difference between the perceptron and Adaline was in the algorithm for weight update (learning). The perceptron had a very crude weight update method which changed neuron weights only if the classification was wrong. In this case, the weight change was calculated as the input value multiplied by the learning rate parameter η. Adaline used a Least–Mean–Square (LMS) algorithm for weight update. The objective of the Adaline learning process could be defined as finding the optimal set of synaptic weights $w_1, \ldots w_n$ to

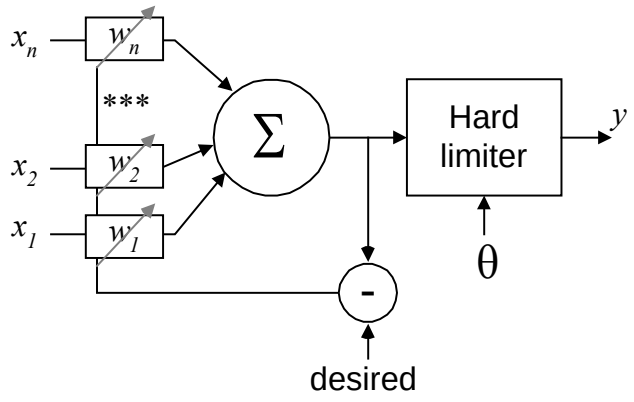

Figure 5.4. Adaline model.

minimize the mean square value of the error *e* for a given set of input patterns and associated desired outputs. Error *e* is calculated as the difference between the actual and desired neuron output. The model of the Adaline neuron is presented in Figure 5.4. The weight update function for the LMS method can be stated as:

$$w_i^{(n+1)} = w_i^{(n)} + \eta(d(n) - y(n))x_i(n),$$

where $w_i^{(k)}$ and $x_i(k)$ are the denoted weights and input signals for the i^{th} input connection in time epoch *k*, and with *d(k)* and *y(k)*, the desired and actual Adaline output in the same epoch.

5.1.2 Networks of neurons

In order to solve more complex problems, perceptrons or Adalines must be integrated into networks. To be effective, neural networks are usually organized into layers. In the simplest form we have one layer of input neurons (input layer) and one layer of output neurons (output layer). Input neurons receive data (signals) from the environment on which some operation (such as classification, prediction or a similar task) should be performed. Output neurons generate the resulting data. The terms *node* and *unit* are also used to reference neurons in the neural networks. Inputs to the network are denoted as $x_1, x_2, \ldots, x_n$ or as a vector $\boldsymbol{X}$. The output from the network is denoted as $y_1, y_2, \ldots, y_m$ or as a vector $\boldsymbol{Y}$. Data transfer and data processing takes place at discrete time intervals (epochs). The input data at time epoch *t* is therefore denoted as $\boldsymbol{X}(t)$.

If all connections and signal transfers start from input neurons and finish in output neurons we have a *feed-forward* type of neural network. Such a network, consisting only of input and output neurons, is classified as a *single-layer feed-forward network*. It is single layer because only neurons in the output layer perform data processing. There is no data processing in the input neurons as they simply transfer input to the neurons in the output layer. A more general topology with one or more intermediate (hidden) layers of perceptrons is known as a *multi–layer feed-forward network* or a *multi–layer perceptron*. A multi–layer perceptron with one hidden layer is presented in Figure 5.5. Input layer neurons (nodes) are shown in white to distinguish them from nodes in hidden and output layers.

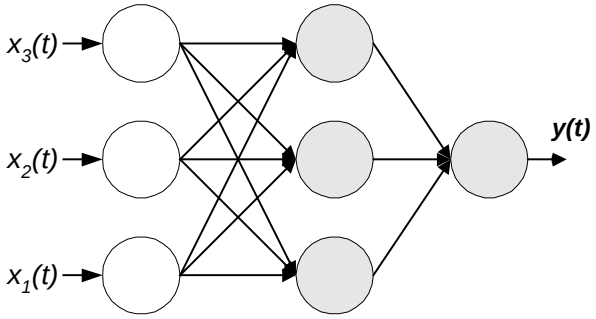

Figure 5.5. Multi-layer perceptron with one hidden layer.

In their study of perceptrons, Minsky and Papert made an intuitive judgement that the limitations of the perceptron will, to a large extent, also limit the multi–layer network of perceptrons [Minsky 1969]. However, later developments proved that multi-layer perceptrons can solve much more complex problems than isolated perceptrons. For example, multi-layer perceptrons have been applied to such complex domains as optical character recognition and speech recognition.

A multi–layer perceptron is a *fully connected* feed-forward network. This means that each neuron from one layer is connected to all the neurons in the next layer. A topology in which some synaptic connections are missing from the network is called *partially connected*. In practice, the design of a partially connected network (through choice of removed connections) reflects prior knowledge about the domain for which the network is constructed. For example, knowledge about the characteristics of the pattern being classified with a given network will influence the decision about which connections to remove from the network.

Feed-forward networks based on Adaline neurons were constructed by Widrow and his students [Widrow, 1962] and were known as *Madalines*. Madalines had many inputs and Adaline neurons in the first layer. The second layer was constructed by using various logic devices such as AND, OR and voting elements. A fixed logical function requested a new design for each problem, which made the use of Madalines much more limited than multi-layer perceptrons.

If the network has at least one feedback connection we call such as architecture a *recurrent* neural network. Feedback is the connection from one neuron output to its own input (in which case we call it self-feedback) or to any other neuron in the previous

layers so that it influences the input applied to that particular neuron. The presence of the feedback connection has a profound impact on the learning capability of the network and its performance. Recurrent neural networks will be elaborated on in more detail in the section about neural network architectures for forecasting.

Another important neural network topology is the *lattice* architecture consisting of a one or multi-dimensional array of neurons. There is also a corresponding set of input nodes which supply input signals to the array of neurons. The neuron position inside a lattice has special significance in neural network training and evaluation. In general, the lattice architecture represents a special case of feed-forward networks.

There are two operations which can be performed on neural networks: evaluation and training (learning). Evaluation is the task of calculating the network output for the present input data. Training is the task of changing the network's free parameters (weights and others) in order to improve its performance. As the number of free parameters could be large, their types different, and their relationship complex, training of neural networks is a serious research problem.

5.1.3 Neural network training by back-propagation

In 1986 Rumelhart and co-authors published a paper on a *back-propagation learning* algorithm [Rumelhart, 1986] which made it possible to train multi–layer perceptron networks. The back-propagation algorithm looks for the minimum of the error function in the weight space using the method of gradient descent. The combination of the weights which minimizes the error function is considered to be a solution of the learning problem. It can be viewed as a generalization of the least-mean-square error algorithm.

This same approach for neural network learning was used (and published) by Werbos [Werbos, 1974] and others. However, the "intellectual climate" then did not allow for the quick acceptance it received in 1986. The development of the back-propagation algorithm proved a real breakthrough in neural network research as it provided a computationally efficient method for the training of multi–layer networks.

The back–propagation algorithm, as other techniques for training neural networks, is an adaptive (re-estimation) method. As represented in Figure 5.6, the learning process starts with random values for the network's free parameters (weights). Training data is

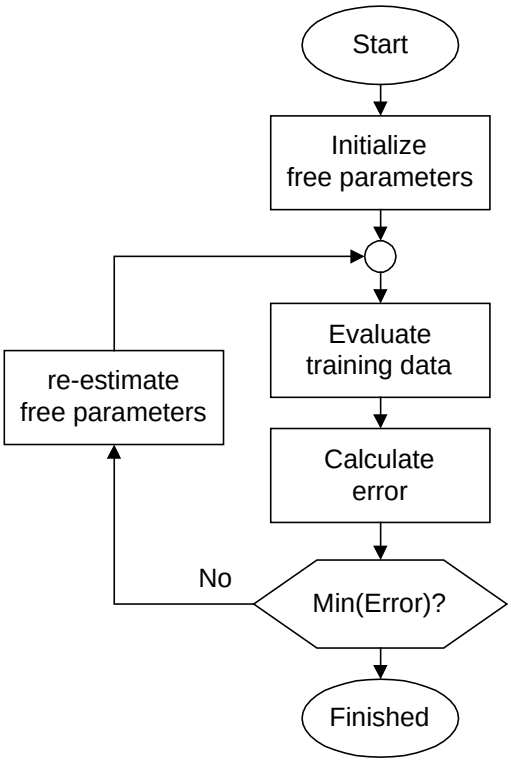

Figure 5.6. Re-estimation learning process.

then presented to the network and the corresponding output is calculated. The difference between the actual and desired output from the network is calculated and referred as 'error". This value is then used to change the network weights in order to minimize deviations between prediction and desired outcomes. The process is repeated until a local minimum for error is reached.

One important requirement for the back-propagation algorithm to work is the continuity and differentiability of the error function (needed in order to calculate the gradient). Consequently, the threshold function used in the standard perceptron is not appropriate and other activation functions have to be used. The most popular functions used in neural networks are the *sigmoid* (logistic) function defined as: $y(s) = \frac{1}{1+e^{-s}}$ and the *hyperbolic tangent* function defined as: $y(s) = \tanh(s) = \frac{1-e^{-2s}}{1+e^{2s}}$. Sigmoid and tanh activation functions are shown in Figure 5.7.

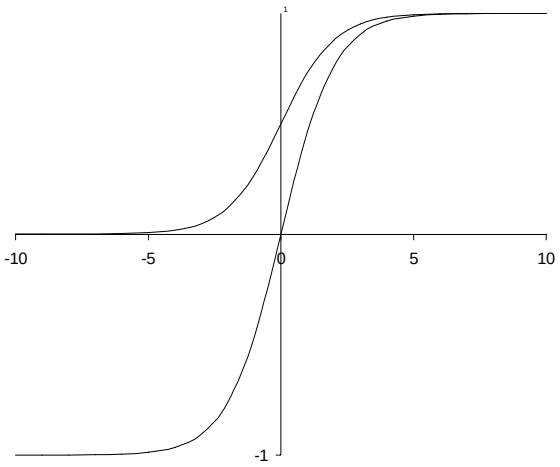

Figure 5.7. *Sigmoid* and *tanh* activation functions.

Back-propagation is a *supervised* learning method, as a correct function output for each input set is required in order to calculate the error. The basic idea of back-propagation is to calculate the network error ($e = d - y$), where d denotes the desired network output and y denotes the actual network output, and propagate the calculated error back through the network to modify weights. The network weight correction is calculated using the following formula (learning or *delta* rule):

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n),$$

where η is the learning rate, $\delta_j(n)$ is the local gradient, and $y_i(n)$ the actual output of the node. The local gradient δ is recursively computed by passing the error signal leftward through the network, layer by layer. Skilful choice of the learning rate constant η is important to allow fast learning while avoiding instability. A simple method of improving the learning vs. stability rate is to add a momentum term to the learning rule:

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n).$$

The *momentum term* ($\alpha \Delta w_{ji}(n-1)$) provides significant improvements as it accelerates descent towards the minimum, avoids oscillation, and in some cases prevents the learning process from terminating in a shallow local minimum. The *momentum constant* α is a small positive number, which controls the influence of the previous weight change in the current calculation.

We should note that the back–propagation algorithm is limited to feed-forward neural networks, and other learning techniques must be used if the network architecture contains recurrent connections.

Research into neural networks' architectures and learning algorithms has been a very active area in the last twenty years. Many new architectures and algorithms have been proposed and neural networks have been used to solve many real-world problems. Reviewing neural network advances is beyond the scope of this thesis and the interested reader is referred to any of the many books on this topic (for example: [Haykin, 1994]). It is interesting that multi-layer preceptrons and the back–propagation learning method, while relatively simple and old, are still the most popular and widely used techniques in many applications.

In the next section we will present an overview of neural network architectures and applications in temporal prediction.

5.2. Neural networks for predictions

The human desire to control life and the environment seems deeply ingrained. This desire led the progress of civilization through technical achievements over the last few thousands of years. Two important tasks for controlling any process are predicting the future and understanding the past. For example, for people who lived a few hundred years ago it was important to predict weather developments in order to ensure a good food production. For some part of modern society it is important to correctly predict share prices. In both cases scientific research can and has been employed to find underlying laws and patterns.

Analysing both the past and future in temporal data can be accomplished using one of two fundamental approaches: finding a "physical law" which governs a particular process, or finding a way to emulate a system's behaviour without knowledge of underlying laws. The first approach can give accurate results, but is not possible in many cases because of the complexity of the system, noise, the stochastic nature of the process, etc.

A temporal process, for which we do not know any exact laws, can alternatively be described by observations taken sequentially, usually at discrete, equally spaced intervals. Such a sequence of data is known as a *time series*. There are many examples of time series in economics, engineering, natural and social sciences and other fields.

The analysis of time-series data has three major goals:

- *Forecasting* (*predicting*) – finding the short-term evolution of the system (assuring the future is like the past),
- *Modelling* – finding the long term behaviour of the system (assuring the future is like the past),
- *Characterization* – determining fundamental properties of the system without a-priori knowledge.

Given an observed time series, the goal of *prediction* is, to predict its future values. Many authors use the term *forecasting* for the same process and in this thesis the terms prediction and forecasting will be used interchangeably. If we denote an observed time series data with $x_1, x_2, \ldots, x_T$, the prediction problem can be formulated as finding y_{T+N}. Data items x_t can be univariate or multivariate, that is, can consist of one or more independent variables. For univariate data $y \equiv x$, while in the case of multivariate data, y could be any non-empty subset of x. The integer N is called the *lead-time* and represents the number of time epochs between the moment when the last time-series data item is collected and the moment at which we would like to know (predict) its value. Modelling and characterization processes are not relevant for this section and will not be further explored.

Forecasting time series is a relatively new scientific area – until the 1920*s* forecasting has been done by simply extrapolating data in the time domain. In 1927 Yule invented the *autoregressive forecasting technique* in order to predict the number of sunspots per year [Yule, 1927]. In this technique the next value is calculated as the weighted sum of the previous observations in the series:

$$\bar{y}_t = f_1 y_{t-1} + f_2 y_{t-2} + \ldots + f_p y_{t-p} + x_t .$$

Here $\bar{y}_t$ denotes the estimated (forecasted) value in the time epoch t, y_k denotes the observed value at time epoch k, and x is the input value.

The autoregressive method represented a big improvement in linear model prediction. Other models have been developed, such as the *moving average* (MA), which calculates the forecasted value as the weighted sum of a number of previous input values: $\bar{y}_t = \sum_{n=0}^{N} b_n x_{t-n}$.

A combination of autoregressive and moving average techniques known as the *ARMA* model has also been developed. For prediction, the ARMA model uses both previous observations and previous predicted values: $\bar{y}_t = \sum_{n=0}^{N} b_n x_{t-n} + \sum_{m=1}^{M} a_m y_{t-m}$.

The major disadvantage of the above methods is that they are limited to linear models only. Since the majority of processes in the nature or industry are non–linear in nature and drastic simplifications can rarely be justified, the application of these methods is often not possible.

An important step beyond linear models was taken in 1980 when Tong and Lim proposed the use of two, instead of one, linear functions [Tong, 1980]. This *threshold autoregressive model* is globally non–linear as it chooses one of two functions based on the system's state. Many other non–linear models followed, including *exponential autoregressive* [Ozaki, 1982], and *autoregressive with conditional heteroscedastisity* [Engle, 1982]. However, non-linear models are more than one order of magnitude more complex than linear models and are difficult to implement. On the other hand, linear models preserved their usability and popularity in some domains such as financial forecasting.

Neural networks have some very convenient features which make them valuable and attractive for a forecasting task [Zhang, 1998]. Firstly, they are a data-driven self-adaptive method that makes them appropriate for situations where knowledge about the underlying process is limited to the case where only data from previous experiences with the system are available. Secondly, generalization capabilities of neural networks enable them to make correct predictions even for previously unseen data as long as the underlying processes remain stable. Finally, neural networks as non–linear, universal function approximators are more flexible than traditional statistical methods that are usually limited to a particular class of functions.

Zhang also surveys research into the relative performance of neural network forecasting compared to statistical forecasting methods. Research reports in the literature are inconsistent, as some researchers reported that neural network performances are better, and others that statistical methods are better. In a complex domain such as temporal forecasting, it is normal that parameter selection and variety in data sets leads to different results. However, the majority of reports prove that neural networks have better prediction capabilities than statistical methods.

5.2.1 Neural network forecasting applications

The first neural network application for forecasting was reported shortly after Rosenblatt's perceptron was proposed. Hu used an adaptive linear network (adaline) for weather forecasting [Hu, 1964]. In his thesis [Werbos, 1974] Werbos found that neural networks with a backpropagation learning algorithm could out–perform traditional statistical methods for forecasting. As with other applications of neural networks, the re-introduction of the backpropagation algorithm in 1986 induced a much faster development. In 1987 Lapedes and Farber [Lapedes, 1987] proved that neural networks can be used to forecast a non–linear time series. They used a feedforward network to predict two deterministic chaotic time series generated by a logistic map and Glass-Mackey equation. In the famous Santa Fe Institute forecasting competition in 1993 [Gershenfeld, 1993] neural network models were winners for each data set, confirming the great prediction performances of neural networks.

During the nineties neural networks became a very popular forecasting tool due to their properties and the wide availability of computers with sufficient power to process time series. Many different applications of forecasting by neural networks have been reported. One recent survey of the state–of–the–art in forecasting with neural networks [Zhang, 1998] lists a number of different neural network forecasting applications. One important group is the forecasting of deterministic chaotic time series from physical phenomena and forecasting the number of sunspots. Forecasting of sunspots has been well-studied in the statistical literature and is used as a benchmark to measure the forecasting performances of various methods, due to their stochastic properties.

The second important group of neural network applications in forecasting is in financial applications: predicting fluctuations in foreign exchange rates, stock prices, commodity prices, bankruptcy and business failure, and others. The third group of applications is in production and manufacturing. For example, electric load consumption forecasting was a research topic for more than ten teams. Among other applications were predictions of environmental temperature, international airline passenger traffic, the ozone level, river flow, tool life, total industrial production, and many others.

One important class of neural network forecasting applications related to intelligent transportation systems will be reviewed in Section 5.3.

5.2.2 Neural network architectures for forecasting

In previous chapters we showed that neural networks have some features which make them very usable for prediction. We also showed that neural networks have been used to solve a large number of different prediction tasks. However, building a neural network forecaster for a particular forecasting problem is not trivial [Zhang, 1998]. One of the reasons is that many different neural network architectures have been proposed for processing time-series data (temporal pattern recognition and other related tasks). In many cases, there is no clear evidence of the advantages and disadvantages of any particular architecture, for a particular processing task.

In this section we will give an overview of neural network topologies proposed for time series processing tasks. Many of these architectures are also used in spatial or spatio-temporal applications. It is not the goal of this section to present a complete list or taxonomy of such networks, but rather to present the basic concepts and most popular architectures. More detailed reviews and taxonomies can be found in [Mozer, 1993], [Horne, 1995], and [Kremer, 2001].

As previously stated, statistical dependence exists between data in a time series. These correlations often are temporal patterns that we would like to classify or predict. In order to exploit such dependence for time series processing, some previous data should be available when a prediction is made. A method to provide this data is to provide some kind of memory in the neural network.

Generally, there are two types of memory in neural networks:

- **Long–term memory** contains information related to the complete set of training data beginning from the first training step. It is represented in a network as various trainable parameters. The most important trainable parameters are synaptic weights. Other types of trainable parameters include transmission delays, connectivity schemes, and initial activations. The set of all trainable parameters is denoted by $\boldsymbol{w}$. Long term memory is updated (learned) during the training. Once a network has been successfully trained, the long–term memory is fixed and used for evaluation (prediction).
- **Short–term memory** contains information which is related to the last few states. As such it cannot be fixed and is re–computed for each new input vector during both training and evaluation. The existence of the short-term memory allows a neural network to deal with temporal patterns in time–series data, as it retains aspects of the input sequence relevant for making a prediction.

In systems with short-term memories, activation at any time t is computed based on activations at time t–1 and/or earlier. The term *state vector* (denoted as *S(t)*) is used to represent the set of activations at time t which affects activations in the next time epoch.

Mathematically we can formulate the previous statement as:

$$S(t) = f_s(S(t\text{-}1), X(t), \boldsymbol{w})$$

$$Y(t) = f_y(S(t), \boldsymbol{w})$$

The physical meaning of these formulae can be expressed by the following statement: the state of the system is calculated first, based on the previous system state, current input vector and a set of trainable parameters. Then, a new output vector is calculated based on the current system state and a set of trainable parameters.

The abstract formulation of a neural network system for time–series prediction can be regarded as consisting of the short–term memory and a generic predictor as shown in Figure 5.8. The relationship between short–term memory and the generic predictor depends on the network architecture. In the simplest case, short-term memory is separated and precedes generic predictor units. However, in most cases the relationship between short-term memory elements and predictor units is much more complex.

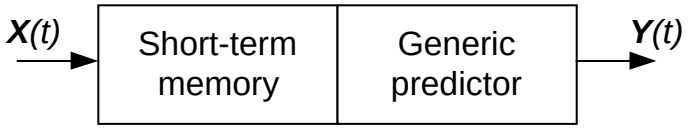

Figure 5.8. Abstract formulation of neural network time series prediction.

Splitting short-term memory from the generic predictor makes it easy to formulate the generic training algorithm for neural networks for temporal prediction. Its pseudo code is shown in Figure 5.9. However, as can be seen later, the implementation of the

```
W= f_wo  (a priori info)          // initialize trainable parameters
do                                // begin training
   t = 0                          // reset time
   do                             // begin current input sequence
      t = t + 1                   // increment time
      x(t) ← environment          // determine input vector
      s(t) ← f_s(s(t-1), x(t), w) // update state
      y(t) ← f_y(s(t), w)         // determine output
      if desired y*(t) is available // if have desired output
         E ← y*(t) - y(t)         // compute error vector
         ε ← ½ ‖E²‖               // compute scalar error
         w ← w + f_Δw(E, w, x(·)) // update trainable parameters
      while not end of sequence   // until end of input sequence
   while t < t_max and ε > threshold // solution found or out of time
```

Figure 5.9. Pseudo code for generic training algorithm [Kremer, 2001].

$\boldsymbol{W}$	the set of network trainable parameters;
$\boldsymbol{f_{wo}}$	the function to calculate initial values for trainable parameters,
t	the discrete time index;
$x(t)$	the input vector;
$s(t)$	the state vector;
$y(t)$	the output vector;
$y^*(t)$	the vector of desired outputs;
E	the error vector;
ε	the total error;
$x(\cdot)$	the input history;
f_s, f_y, and $f_{\Delta w}$	the functions which calculate the system state, output and new values for trainable parameters, respectively.

training algorithm is not trivial for a number of reasons. The major problem lies in updating trainable parameters.

There are many different neural network architectures for forecasting. In the following subsections some of the most important of these will be presented.

Window In Time multi-layer perceptron

Multi-layer perceptrons, as the most popular neural network architecture, have often been used in forecasting applications. This architecture is based on the simple idea of extending a multi-layer perceptron to use input vectors from a few previous time epochs. Such "short time memory" opens a "Window–In–Time" on the input vectors. The best–known implementation of Window–In–Time networks was the NETtalk system [Sejnovski, 1986], but it has also been used by other researchers. The Window-In-Time architecture is presented in Figure 5.10.

The state vector of the Window–In–Time network consists of n previous input vectors:

$$s(t) = x(t) \oplus x(t\text{-}1) \oplus \ldots \oplus x(t\text{-}n+1),$$

where the operation $\oplus$ is a concatenation of sub–vectors in a vector. According to the previous definition, the state space is recursively calculated using the following formula:

$$f_s(s(t\text{-}1)), x(t), \boldsymbol{w}) = x(t) \oplus s(t\text{-}1)[1..(n\text{-}1)]$$

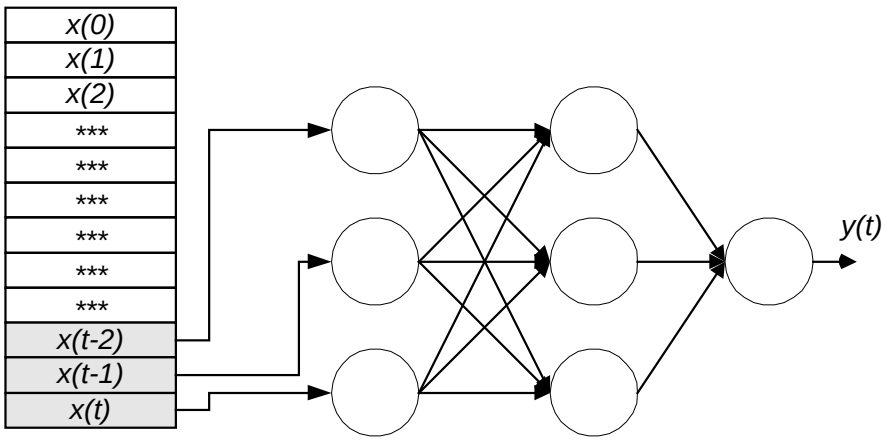

Figure 5.10. Window-In-Time architecture.

where the operation $a[1..k]$ is defined as the extraction of a sub–vector of vector a, consisting of elements 1 to k.

The general predictor part of this network is an unmodified multi–line perceptron, and a standard back–propagation training algorithm can be used. Window–In–Time networks are well suited for non–linear prediction on a *stationary* time series. A time series is stationary when its statistics do not change over time.

Time–Delay Neural Networks

Time-Delay Neural Networks have been developed by a group of researchers at Carnegie Mellon University (working in cooperation with researchers in other institutions) in the late eighties [Waibel, 1989; Lang, 1990]. Time-Delay Neural Networks (TDNN) can be thought of as an extension of multi–layer perceptrons with delay lines that provide the history of the input data for each processing node. In Figure 5.11. a model of a neuron with delay lines applied to its inputs is presented.

Consider that there are J nodes in the previous layer. There is one direct connection from each node in the previous layer, plus N connections that are connected through delay lines D_1 to D_N. A delay line D_k propagates input with a delay of k time epochs. Each of the J inputs into the node should be multiplied by several weights, one

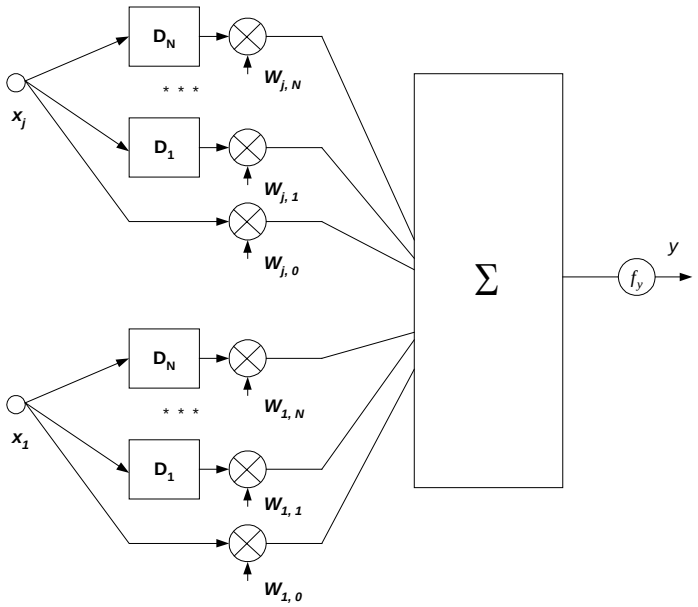

Figure 5.11. Time-delay computation unit.

for each delay plus one for the undelayed input. Consequently, the unit will have *J*(N+1)* weights. If we denote weights for input *i* by $w_{i,k}$, $k=0,\ldots,N$, the output from the unit can be calculated using the following formula:

$$y = f_y\left(\sum_{i=1}^{J}\sum_{k=0}^{N} w_{i,k}x_i\right)$$

Nodes with delay lines could be included only in the first hidden layer of the neural network, in which case we call this a *focused* topology, or they could be included in all hidden and output layers. Focused time-delay networks are functionally equivalent to the Window–In–Time architecture. Figure 5.12. presents a block diagram of a time–delay neural network (TDNN).

Extending multi–layer perceptron by adding delay lines allowed most existing techniques developed for feed–forward neural networks to be easily used in implementations. The back–propagation learning method is favoured by many TDNN. However, a problem with TDNN is that there are a large number of weights and that learning can therefore be very slow.

A state vector for a TDNN consists of a number of input vectors from previous

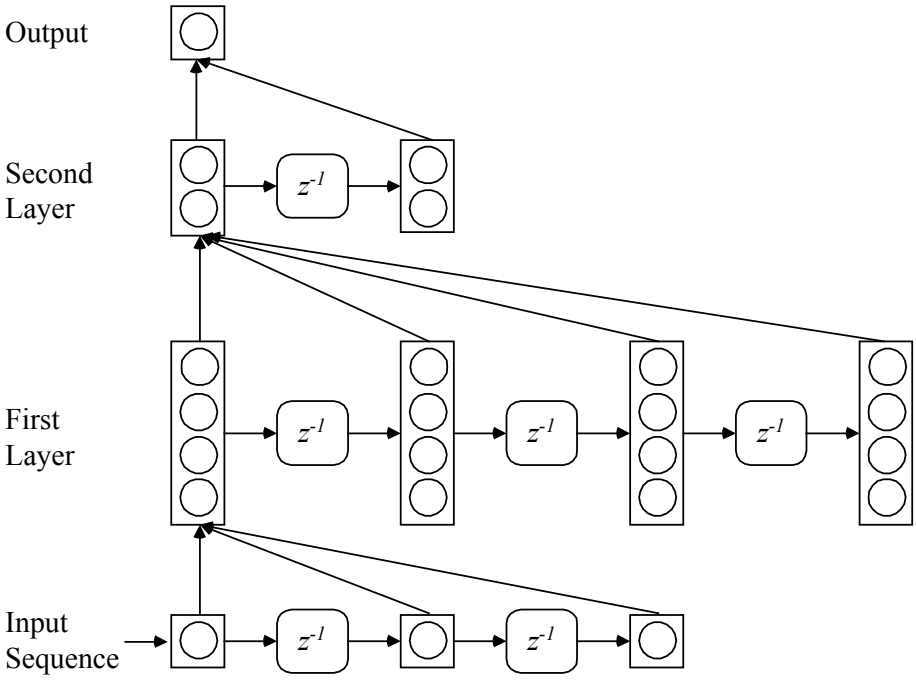

Figure 5.12. Example of Time–Delay Neural Network.

epochs and a number of activation vectors from each of the hidden layers, again for each of a predefined number of time epochs:

$$s(t) = x(t) \oplus x(t\text{-}1) \oplus ... \oplus x(t\text{-}n\text{+}1) \oplus h^{I}(t) \oplus h^{I}(t\text{-}1) \oplus ... \oplus h^{I}(t\text{-}m\text{+}1) \oplus h^{II}(t) \oplus h^{II}(t\text{-}1) \oplus ... \oplus h^{II}(t\text{-}l\text{+}1)$$

This formula is valid for a network with two hidden layers with n delay lines in the input layer, m in the first and l in the second hidden layer. Activations of the first and second hidden layers are denoted by h^{I} and h^{II} respectively.

Instead of using delay lines, the TDNN could be represented as "unfolded" into a static structure, multiplying nodes and connections to represent delay lines. The unfolded version of a TDNN network is convenient to understand the complexity of the system if standard algorithms are used for back–propagation learning. The simplest training algorithm for TDNN is based on the idea that for each recurrent network (or network with memory elements) it is possible to construct a feed–forward network with behaviour which is identical for a given time interval. This algorithm is known as *back-propagation through time* and was first described in [Werbos, 1974]. A serious limitation of this algorithm is its computational feasibility due to the complexity of the "unfolded" network.

A functionally equivalent network has been proposed by Wan [Wan, 1993], called the *Finite Impulse Response* (FIR) network as presented in Figure 5.13. Wan's FIR

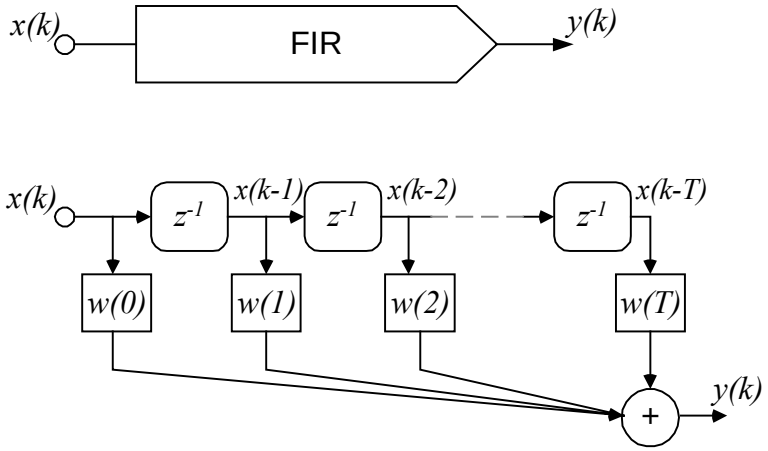

Figure 5.13. The Finite Impulse Response model.

network won the Santa Fe Institute forecasting competition, proving its superior forecasting performances. The idea was to replace all connections in a feed–forward network with Finite Impulse Response (FIR) filters that provide short-term memory for the network. FIR filters are implemented as a series of delay lines, making the network equivalent to a TDNN. In order to make learning computationally feasible, Wan proposed a temporal back-propagation algorithm. The main idea of temporal back-propagation is to use FIR filters to perform inverse filtering on a local gradient δ. This is consistent with back-propagation and provides for a more computationally efficient method for updating weights.

A delay-line is the simplest memory structure that can be used to represent the past in a TDNN. More complex memory structures could compose output as the weighted sum of inputs in the previous time epochs. A simple formalism which characterize these types of memory could be represented by:

$$\bar{x}_i(t) = \sum_{\tau=1}^{t} c_i(t-\tau)x(\tau)\,.$$

In this formula $c_i(t)$ is called a *kernel function* and represents the weight for a specific time epoch. For a simple delay line with one delay element, the kernel function is defined as:

$$ci(t) = \begin{cases} 1, & \text{if } t = \omega_i \\ 0, & \text{otherwise} \end{cases}$$, where ω_i is the delay of the line.

Figure 5.14. presents kernel functions for the time delay and three more complex memory structures: exponential decay, gamma, and Gaussian. These memory structures compose output as the weighted sum of inputs in the previous time epochs. This is a great improvement over delay lines, which output the input signal from only one of the previous time epochs. These structures can implicitly store an infinite history of previous activations. As a consequence, a smaller number of memory structures are required by the network. However, in the case of more complex memory structures, their parameters must be included in the learning process. The back–propagation learning algorithm can therefore not be used in such networks and more complex learning rules must be employed.

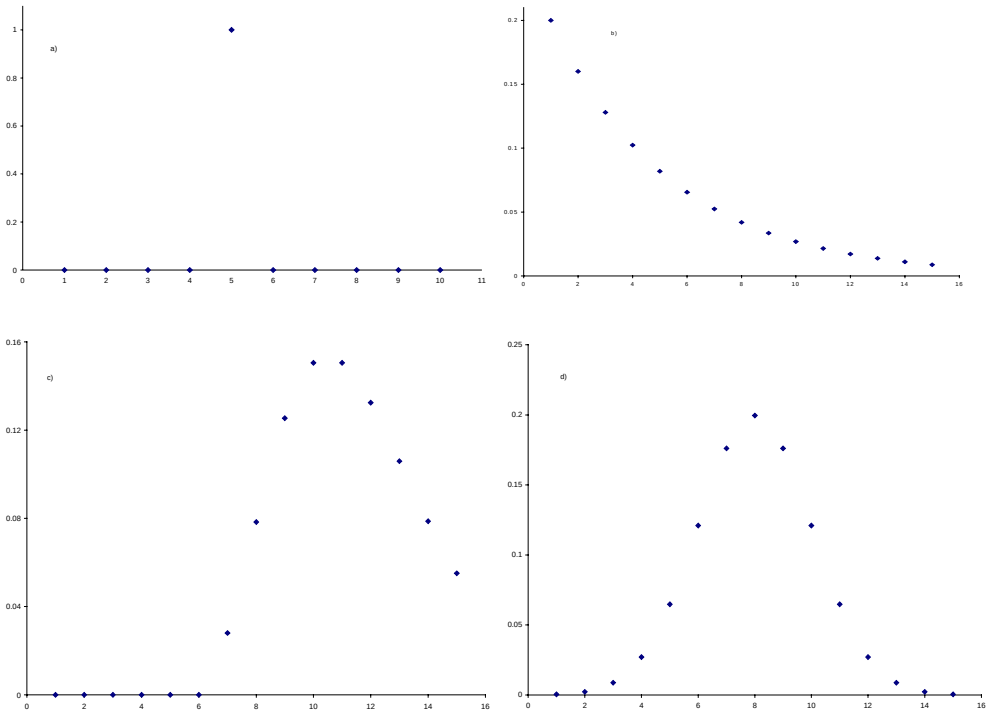

Figure 5.14. Various types of kernel functions exploited as memory nodes in temporal neural networks:

a) Time delay N = 5,
b) Exponential decay memory, $\mu = 0.8$.
c) Gamma memory $\omega = 6$ and $\mu = 0.4$
d) Gaussian memory

Recurrent neural networks for forecasting

Another way of providing a memory–like structure inside a neural network is to make connections which feed back output from a neuron to itself, or to some other neuron in the same or previous layers. This way the network keeps information from previous time epochs and uses it to predict some future values. Neural networks which contain feedback connections are commonly known as *recurrent networks*.

There are two major classes of recurrent networks:

- *Static recurrent networks*, which could be represented as an extension of feed–forward networks, with an added set of context units. Context units are used to memorize past states of units in the hidden or output layers. The feed–forward connections are trainable, while feedback connections which transfer

data to the context units are not trainable (are fixed). Context units possess characteristics of dynamic memory.

- *Adaptive recurrent networks*, which may have arbitrary feed–forward and feedback connections, all of which are trainable.

Adaptive recurrent networks have greater flexibility than static. However, training of static recurrent networks is much easier, as simple and well-known algorithms developed for feed–forward networks (such as the back–propagation algorithm) can be used. Examples of static recurrent networks are Elman, Jordan and NARX networks, described in the following text.

Non-linear Auto-Regressive network with Exogenous Inputs (NARX)

Non-linear Auto-Regressive networks with Exogenous Inputs were first described in [Chen, 1990] and [Narendra, 1990]. They represent one of the simplest recurrent architectures and can also be regarded as an extension of Window-In-Time networks (which represent a special case of the NARX architecture). However, recently it has been demonstrated that NARX networks are computationally equivalent to Turing machines [Siegelmann, 1997] and that they can easily learn long-term dependencies [Lin, 1996].

A NARX architecture represents a standard feed–forward network consisting of an input layer, one hidden layer, and an output layer. Its input consists of a normal input

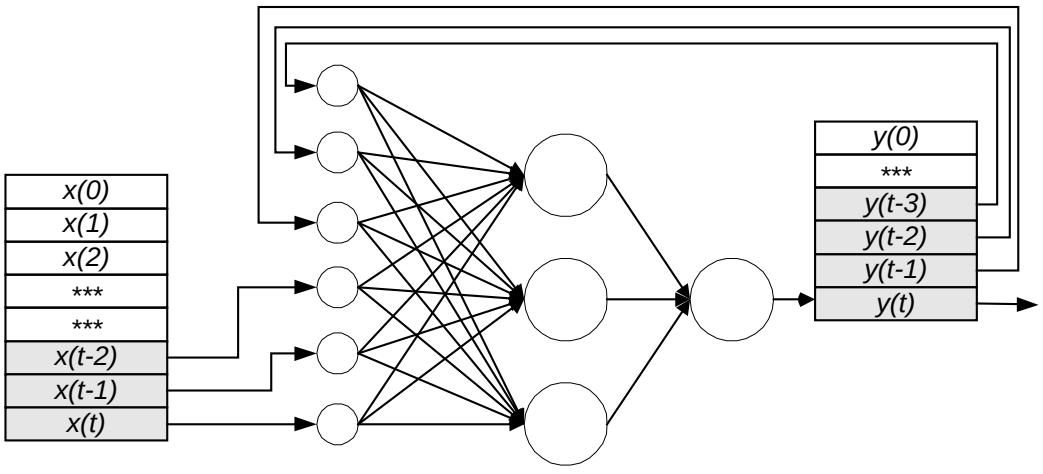

Figure 5.15. Non-linear Auto-Regressive network with Exogenous Inputs (NARX).

vector, a number of previous input vectors (as in the WIT architecture), and a number of previous network outputs. Figure 5.15. represents a NARX architecture. The state vector for a NARX architecture can be expressed as:

$$s(t) = \{x(t) \oplus x(t\text{-}1) \oplus ... \oplus x(t\text{-}n\text{+}1)\}$$
$$\oplus \{y(t\text{-}1) \oplus y(t\text{-}2) \oplus ... \oplus y(t\text{-}m)\}$$

Elman Networks

The Elman network was introduced in [Elman, 1990] and consists of (Figure 5.16):

- One or more input units, where data from an input vector *x(t)* are applied.
- A number of hidden units which can have linear or non–linear activation functions.
- One or more output nodes.
- A number of context units, which memorize the previous activation of hidden units.

At a specific time *t*, the previous activation of the hidden units (from time *t*–1) and the current inputs (at time *t*) are used as inputs for the network. These inputs are propagated forwards to produce the outputs. The standard back–propagation learning algorithm is then employed to train the network. After each training step, activations of hidden units are sent back through recurrent links to context units to prepare input for

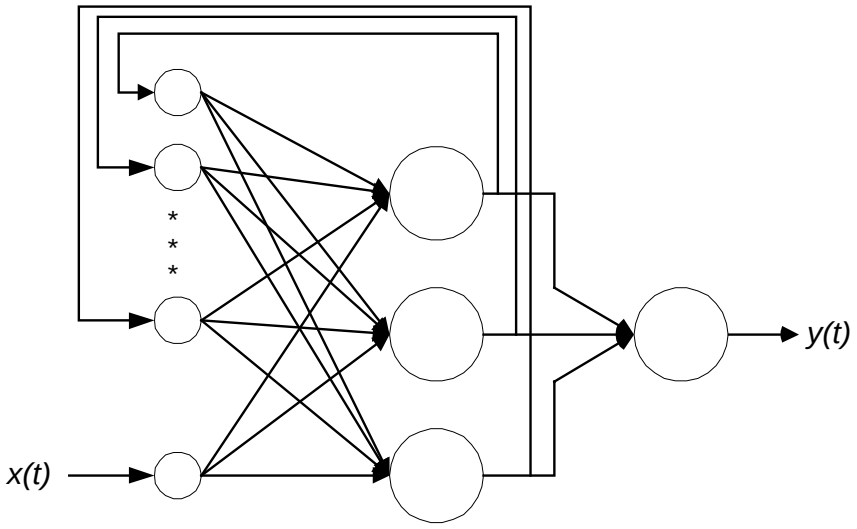

Figure 5.16.. Elman network.

the next training or evaluation step. At the beginning, activations of hidden units are not known and they are usually preset to a value which is in the middle of the input range for the activation function. Short term memory in Elman networks is stored in a set of hidden units and can be formulated as:

$$s(t) = f_s(s(t-1)), x(t), \boldsymbol{w}) = \sigma(\mathbf{W} \cdot \{1 \oplus x(t) \oplus s(t-1)\})$$

where $\mathbf{W}$ is a matrix representing a subset of the set of all trainable parameters $\boldsymbol{w}$, while $\sigma()$ represents the selected activation function for the network. The state vector can be easily expressed as:

$$s(t) = ½s(t-1) + y(t-1).$$

Theoretically, an Elman network can model an n^{th} -order dynamic system, where n is the number of units in the hidden layer. However, experiments found that Elman networks can identify first order linear systems easily, but have difficulties in identifying linear systems of higher order. A number of extensions in architecture and learning techniques have been proposed to overcome this problem.

Jordan Networks

Jordan networks [Jordan, 1986] use the current input signal and previous output values as inputs to the network, as presented in Figure 5.17. It is a classical feed-forward architecture that consists of one or more units in the input, hidden and output layers. This is extended by a number of context units, which are connected to outputs by feedback connections. An important feature of context units in Jordan networks are self-feedback (local loop) connections, which connect the output of context units to the input of the same unit. Self-feedback connections provide context units with longer term memory capabilities, as the output of the context unit could be regarded as a weighted sum of previous network outputs. In mathematical terms, the state of the context nodes can be formulated as:

$$c(k) = \mathbf{\Phi}\, c(k-1) + \mathbf{\Psi}\, y(k-1),$$

where $c(k)$ is a vector of the context units' states at time epochs k, $y(k)$ is the network output at time k, $\mathbf{\Phi}$ is an $n \times n$ matrix (if n is a dimension of c) and $\mathbf{\Psi}$ is an $n \times m$ matrix (if m is a dimension of y).

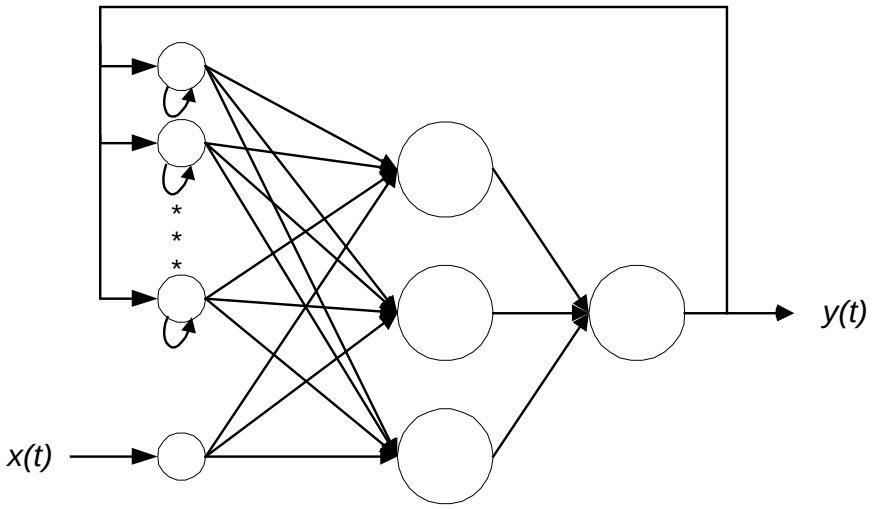

Figure 5.17. Jordan network.

Similarly to Elman networks, because only feed-forward connections are trainable and feedback connections have a fixed weight, these networks can be trained with standard feed-forward training algorithms.

Adaptive Recurrent Networks

Adaptive recurrent networks are also known as general recurrent networks. The advantage of an adaptive recurrent network is in the potentially unlimited memory depth, which enables the full dynamics of the system which produced a temporal signal to be captured. In TDNN memory, depth could also be adaptive; however, due to loss of resolution, this has an effective upper limit.

Feedback connections in adaptive recurrent networks can be implemented in many different ways. One approach, known as *Local Feedback* and described in [Gori, 1989] is presented in Figure 14.18. Local feedback is established by a creating local feedback connection for each node in the first hidden layer. In this case the state vector can be calculated as:

$$s(t) = f_s(s(t-1)), x(t), \boldsymbol{w}) = \sigma(\mathbf{W} \cdot x(t)) + \mathbf{X} \cdot s(t-1))$$

Here both **W** and **X** are trainable parameters and parts of $\boldsymbol{w}$, while **X** is a matrix with zero elements in non-diagonal positions.

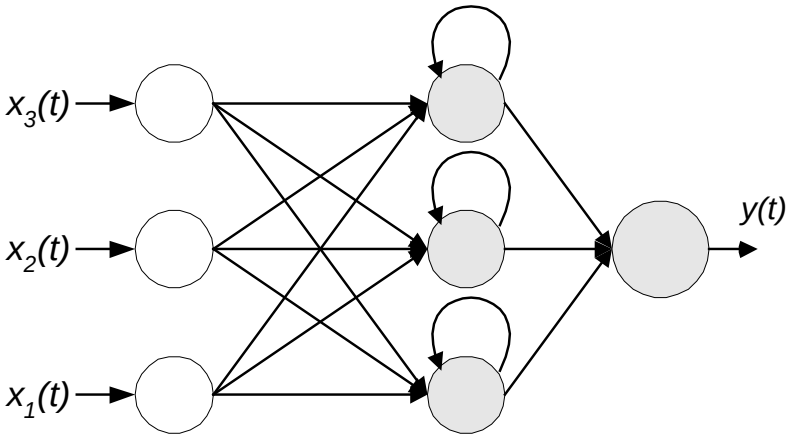

Figure 5.18. Adaptive recurrent network.

There are two general types of adaptive recurrent networks: fully and partially connected. Fully connected recurrent networks have feedback connections for all nodes in the first hidden layer. The local feedback topology, described above, is an example of a fully connected network. Partially connected recurrent networks have the same feedback structure as fully connected networks, but may also have connections which skip the feedback layer.

The possibility of training feedback weights provides additional possibilities for this type of network. However, it is also a source of some disadvantages. The main problem with adaptable recurrent networks is that they can be unstable even for constant inputs. On the other hand, more complex learning algorithms must be used, such as back-propagation through time. However, [Bengio, 1994] showed that adaptive recurrent networks have difficulties learning long-term dependencies because the gradient over time in the *back-propagation through time* algorithm decays exponentially.

5.3. Neural networks in intelligent vehicles

Very good prediction and classification performances of neural networks quickly attracted the attention of automobile manufacturers and researchers in new vehicle technologies ("intelligent" vehicles). In 1993 the first survey paper was published

describing an early application of neural networks to vehicles' dynamics and control [El-Gindy, 1993]. A more comprehensive survey of more than thirty neural network and fuzzy logic applications to vehicle systems is given in [Ghazizadeh, 1997]. Applications are categorized as motion control, driver modelling, tire modelling, braking, suspension, steering, transmission and engine control. Driving modelling is the characterization of the closed loop system of driver, vehicle and environment. In this section we present some relevant research results in this area.

Ghazizadeh and co-authors [Ghazizadeh, 1996] employed a three-layer recurrent network to emulate the dynamic behaviour of a quasi-static vehicle model. The model predicted yaw rate, lateral acceleration and load transfer rates (quasi-static) at the front and rear vehicle axles. The inputs into the model are vehicle velocity, steering angle and the four wheels' vertical loads. Recurrent connections are made by returning output signals from the last two time epochs back to the input layer. Such an architecture could be classified as a non-linear auto-regressive (NARX) network, as described in the previous section. The input layer has fourteen nodes in total. Two hidden layers have five and ten nodes respectively, and the output layer has four nodes. The architecture of the system is presented in Figure 5.19.

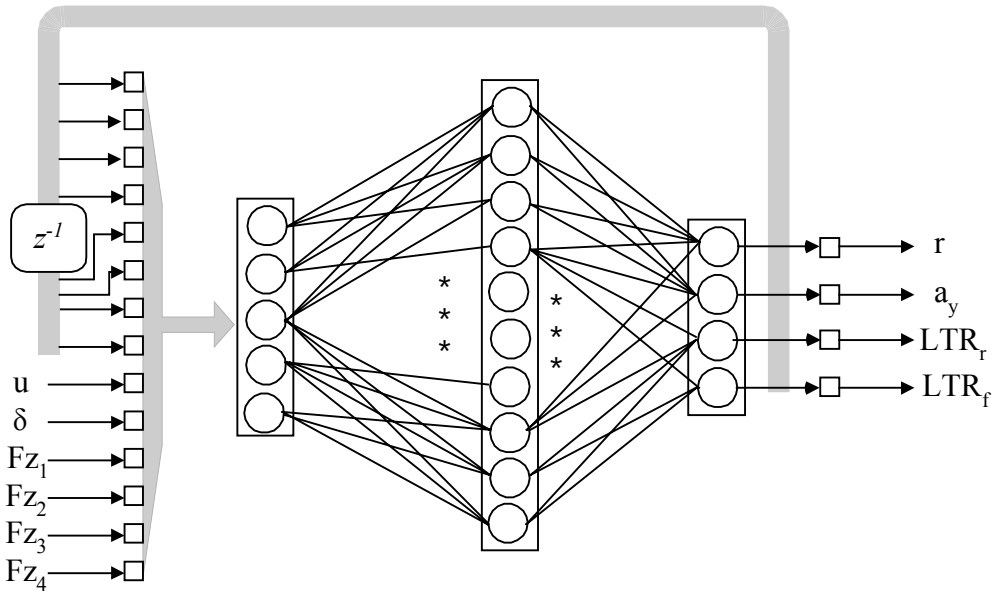

Figure 5.19. NARX network as described in [Ghazizadeh, 1996].

A training data set has been generated by running the vehicle model for three sharp line change evasive manoeuvres. Manoeuvres lasted ten seconds each and were generated for forward speeds of 10, 70 and 130 km/h, to cover the widest range of operating conditions. The test data set was generated by running the same model for speeds of 40 and 100 km/h. The overall error for constant forward speed above 70 km/h was reported to be less than 5%. Authors also reported problems with an accumulation of error which is particularly large when the vehicle starts from the stop condition.

[Kraiss, 1990] used two separate neural networks to emulate a vehicle's directional control and path planning. Directional control is modelled as a function of the steering angle for a given direction error angle and the distance from the next gate to be passed through. Path planning is a choice of the next gate to cross in an artificial environment created as a number of parallel walls with two gates in each wall. For path planning, the network has inputs as distances between doors in the consecutive walls for the next three walls from the current position (Figure 5.20.). The total number of input parameters for a network is twelve and it produces one output (selected gate in the next wall).

The same authors [Kraiss, 1993] presented a driving assistant system to support overtaking. Input parameters of the system are distances to collision on the left and right lines, the lateral position of the vehicle, and its speed. Output parameters are longitudinal guidance (gas/brake pedal position) and lateral guidance (steering wheel

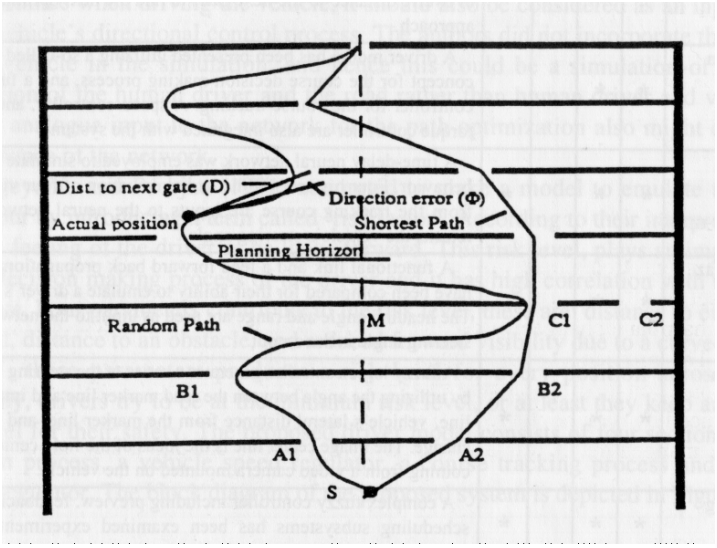

Figure 5.20. Path planning by a neural network (from [Kraiss, 1990]).

position). Authors compared a three-layer feed-forward network, a functional link network and the combination of the two, which, according to the authors, gave the best results (fast learning as functional link networks and good performances as feed-forward networks).

Griswold with colleagues [Griswold, 1994] developed a driver command generator for autonomous vehicle following. Autonomous vehicle following was a popular research domain which will, in the near future, become part of the standard vehicle equipment known as "intelligent cruise control". This feature will enable a vehicle to follow a vehicle in front of it (lead vehicle) at a preset distance.

Griswold's team used two separate two-layer time-delayed neural networks trained to control vehicle speed and steering angle. The network for speed control has 21 nodes in the hidden layer and its inputs are the normalized current vehicle speed, and current and five previous distances to the lead vehicle. The output layer of this network has four nodes with binary output values which represent speed changes of –2, -1, 0 and +1 miles per hour. The network designed for steering control has twelve nodes in the hidden layer. Inputs for this network are the current distance to the lead vehicle, and the current and two previous heading angles (bearing angles to the lead vehicle). The output layer is again implemented as a set of binary nodes (fifteen) corresponding to steering wheel angles of -45, -20, -15, -10, -6, -3, 0, 3, 6, 10, 15, 20, and 45 degrees. The results reported proved that the output from the network is generally acceptable, even if it is different from values generated by human drivers.

A number of systems have been reported which combine machine vision and neural network learning. The most prominent is certainly Pomerlau's ALVINN system [Pomerleau, 1993], claimed to be "reliably driving vehicles at highway speeds for long distances under modest conditions". Developed at the Carnegie Mellon University Robotic Institute, ALVINN is a system for generating steering commands based on incoming images from a video camera and/or laser range-finder. Such systems are usually classified as line keeping or line departure warning systems, since their goal is to keep a vehicle inside given lines and avoid incidents caused by a departure from a given line.

ALVINN is based on a two-layer feed-forward neural network. An image from a video camera or rangefinder is projected on the "retina", consisting of 960 input nodes

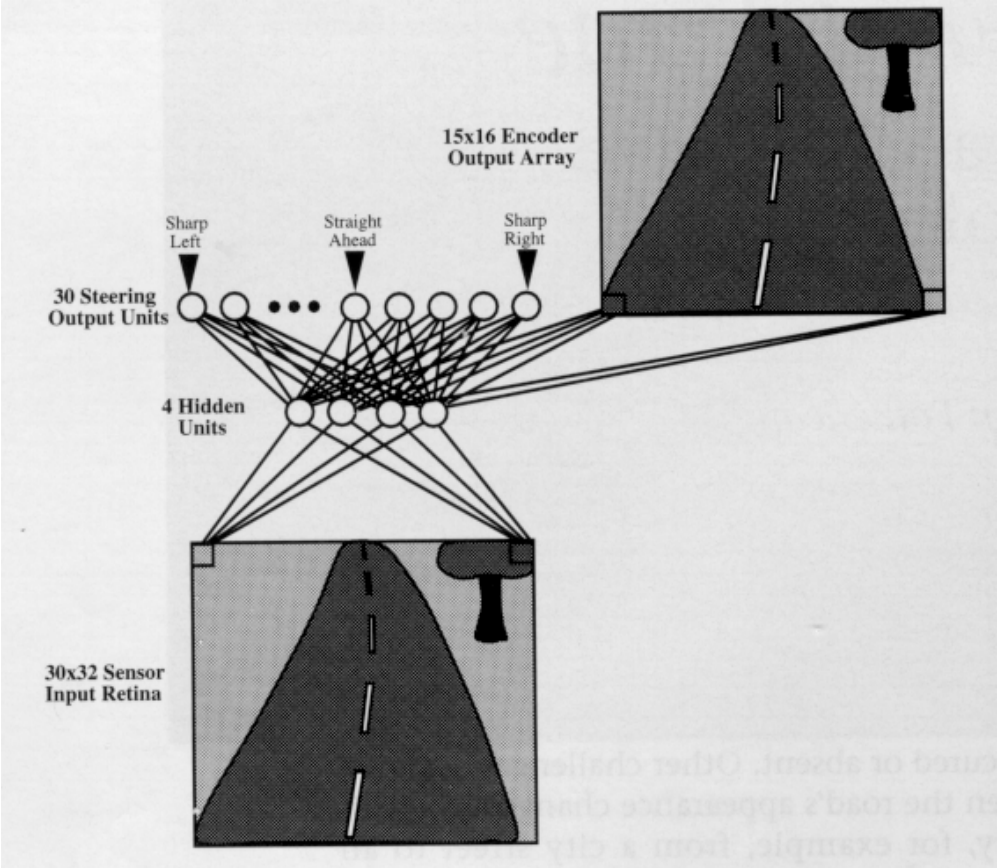

Figure 5.21. Architecture of the ALVINN system [Pomerleau, 1993].

arranged as an array of 30x32 nodes. Input nodes are fully connected to a single hidden-layer of four units, which are further fully connected to an output layer with 30 nodes. Each of the output nodes represents a value for steering curvature. The centremost output node represents the straight-ahead condition, while the left and right nodes represent left and right turns. After propagating activation through the network, the output layer's activation profile is translated into the vehicle steering command. However, the network is not trained to identify one output node with the highest activation. The output is given as a Gaussian activation level centred on the correct steering curvature. The architecture of the ALVINN system is presented in Figure 5.21.

An additional encoder array is also part of the system, representing a second neural network. The purpose of this second array is to provide an estimate of the network's confidence. The network's confidence metric is based on the familiarity of the

input image to the network and is calculated as the correlation between the encoded and input image.

ALVINN does not take into account vehicle driving history and does not try to model the vehicle's position on the road. Instead, it is a simple reflex system for road line keeping. The major advantage of ALVINN over other similar systems is its training algorithm. In general, ALVINN's neural network is trained "on-the-fly": a human driver drives the vehicle, and sensor (video camera) signals are brought as network inputs. At the same time, the actual steering curvature is measured, and the Gaussian distribution is calculated and applied to the network's training algorithm as the desired output. A standard back-propagation learning algorithm is used.

The data set provided by the human driver is very limited as he/she normally will drive very close to the centre of the road. However, for a number of reasons, an autonomous vehicle can easily get into a position which is partially or totally off the road. If the system is not trained with the appropriate data, it will not be able to recover its position. In order to expand the data set, and provide training data which are not provided by the human driver, a set of artificial images is created by shifting and rotating the original image. As illustrated in Figure 5.22. a set of fourteen additional images is generated from the image captured by the camera. The desired output for network training for additional images is calculated using a simple pure-pursuit steering model [Wallace, 1985]. This model calculates the transformed steering angle which will bring the vehicle to a desired location (centre of the road) after some distance. The

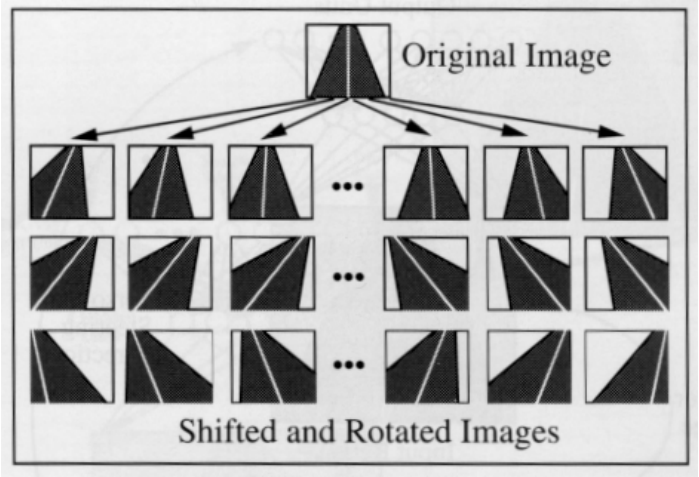

Figure 5.22. Extended training images created in ALVINN [Pomerleau, 1993].

steering transformation is based on the shift and rotation parameters applied on the image transformation. An advantage of this model is that it is independent of the driving situation.

The other important training improvement for ALVINN is its training buffer, which keeps a representative set of input images and uses them to train the network after each new image is captured. Buffering provides the necessary diversity in the training set while preventing training bias towards particular steering directions. The advantages in using transformed and buffered images have been experimentally proven.

ALVINN has been used by a number of different institutions in a number of different vehicles, and it has autonomously driven thousands of miles - including high-speed drives on highways. Subsequently, ALVINN has been extended to support elements of tactical driving, such as line changes and intersection negotiation, while preserving robust line keeping performance [Jochem, 1996].

Owen et al. [Owen, 1996] present a sub-symbolic navigation system for an autonomous mobile robot based on Self-Organizing Feature Maps (SOFM) neural networks. The Self-Organizing Feature Mapping algorithm was developed by Kohonen [Kohonen, 1982]. It is an unsupervised learning algorithm that transforms the incoming signal pattern of arbitrary dimensions into a one or two dimensional discrete map while preserving the topological order. A SOFM network is of lattice type, with the number of input nodes being equal to the dimensionality of the input vector and one- or two-dimensional arrays of output nodes. After applying the input vector, one output node is selected as the winner (best match), and weights for the winner node and its neighbours are updated.

In the system developed by Owen and co-authors, the input vector to the SOFM neural network consists of a sensor input part and an action part. During the training phase, action and sensory information are combined, and the winner's weights and those of its neighbours are updated. The array of SOFM's output nodes represents a very simplified topological map of the environment. During the evaluation (testing) phase, sensory information alone is fed into the network (zeros for action) and an action is retrieved from the weights of the winner.

5.4. Predicting vehicle movement

In the context of the framework proposed in Chapter 3, the predicted vehicle movement (longitudinal and lateral accelerations) will be used to detect unexpected situations. This would happen by comparing predicted and actual acceleration values. The situation model in this case may consists of a few recent instances of actual and predicted values which may provide a way to calculate how much confidence we have in a current prediction. In a situation where a reasonable level of confidence in predictions is established, and a sudden large deviation from the usual behaviour is encountered as a large difference between the predicted and actual acceleration, we may generate a warning for the driver. In such a model of operation, a reasonable level of prediction accuracy is critical. The experiments presented in this Chapter are designed to explore the prediction capabilities of neural networks.

We can envisage a system where the use of short-term predictions will be integrated with higher level models. In such a system, comparisons of short-term predictions and actual data from sensors will be made in the context of a situation model built on a higher level. Such environment may simplify the prediction task, as training and prediction of a network will be restricted only to a specific context. For example, if a higher-level model predicts that the next action is a right turn, then the short-term prediction can be executed on a network which is trained to predict accelerations in right turns only. However, implementation of a such system is outside of scope of this thesis.

5.4.1 The goal of our experiments

The first goal of our experiments with neural networks was to prove that short-term vehicle movement prediction is possible using a very limited set of sensors and well-known machine learning techniques. In our framework, short-term predictions could be used to help the driver to conduct immediate reactive navigation tasks, such as stopping in front of an unexpected obstacle. However, combined with longer term predictions, they could also be used to provide support for more complex navigation tasks.

The second goal of these experiments is to analyse the prediction performances of neural networks for time-series data from standard vehicle sensors. We could use a more diverse set of sensors and dedicated electronics to improve signals and avoid problems such as temperature dependence of the accelerometer's output. However, we decided to keep the set of sensors as small as possible, limiting ourselves only to sensors that are standard equipment in cars now in mass-production. As the result, we decided to conduct the neural network experiments using only accelerometers. Accelerometers are common in modern car electronics even though they are usually not used for navigation, but rather serve as crash sensors to fire air-bags.

The other important goal of our neural network experiments is to compare various neural architectures and their prediction capabilities. We chose to experiment with simple networks and to compare them with more complex architectures that are usually recommended for time-series predictions: time-delayed neural networks and recurrent networks. The "vanilla" neural network architecture as defined by [Moody, 1998] is a two-layer feed-forward network commonly referred to as a multi-layer perceptron (MLP). It is known that a two-layer network can approximate any structure in a given data set [Hornik, 1991].

In previous sections we have argued that neural networks are the machine learning technique of choice for short-term predictions on time series. We also presented some applications of predictions by neural networks in vehicle systems. In the remainder of this chapter we present our experiments on neural networks. In the first section, the data used for the experiments are described. Then we present the software tools used for the experiments, particularly the neural network simulator, *NeuroSolutions*. The third section describes the mathematical model for our experiments, and the remaining sections present our results, as well as their discussion and a conclusion.

5.4.2 Data used for experiments

Before we conducted experiments with neural networks, we designed and implemented a data acquisition system as described in Chapter 4. During and after the implementation of the data acquisition system, a large number of test drives were conducted. Some of these drives were done to determine the validity of the data acquisition hardware and

software, and to determine the parameters for pre-processing (averaging and filtering) of data. Also, a large amount of data was collected while testing the data acquisition system under various conditions. All these test drives form the corpus of data for our experiments.

From these test drives we selected one data set for our neural network experiments. The criteria for drive selection were:

- The test drive should contain a large number of data items so that it can be split into training, testing and cross-validation subsets, each of them large enough to provide effective training, testing and validation.
- The test drive should contain various driving features (starts, stops, left and right turns, etc.) and represent a typical suburban drive pattern in a medium sized New Zealand town.
- The test drive should not contain very long sequences with the same (or similar) values. Iterative re-estimation learning in neural networks causes the network's long-term memory (represented by weights in the processing nodes) to "forget" some input events which happened too long ago. This problem is known and reported in the context of the ALVINN system described in Section 5.3.

The test drive selected from these criteria is shown in Figure 5.23. This diagram presents a test drive through a series of two-dimensional coordinates collected by the GPS receiver in the data acquisition system in parallel with data from sensors. Coordinates have numerical values expressed in the New Zealand Map Grid coordinate system, effectively showing distance in metres from the predefined coordinate system's origin. As the GPS receiver provides position coordinates only in a selected geodetic system (specified as latitude and longitude in degrees), the necessary conversion to the New Zealand Map Grid coordinate system was performed via post-processing software. The selected test drive was conducted before the Selective Availability for Global positioning system was removed and the position accuracy for the data presented here is therefore above 20 metres.

For illustrative purposes we label the first seven driving events with numbers 1 to 7. The first event represents the vehicle start. Events 2 and 5 represent left turns, while 3, 4, 6 and 7 represent right turns. During this test we drove more than once along the

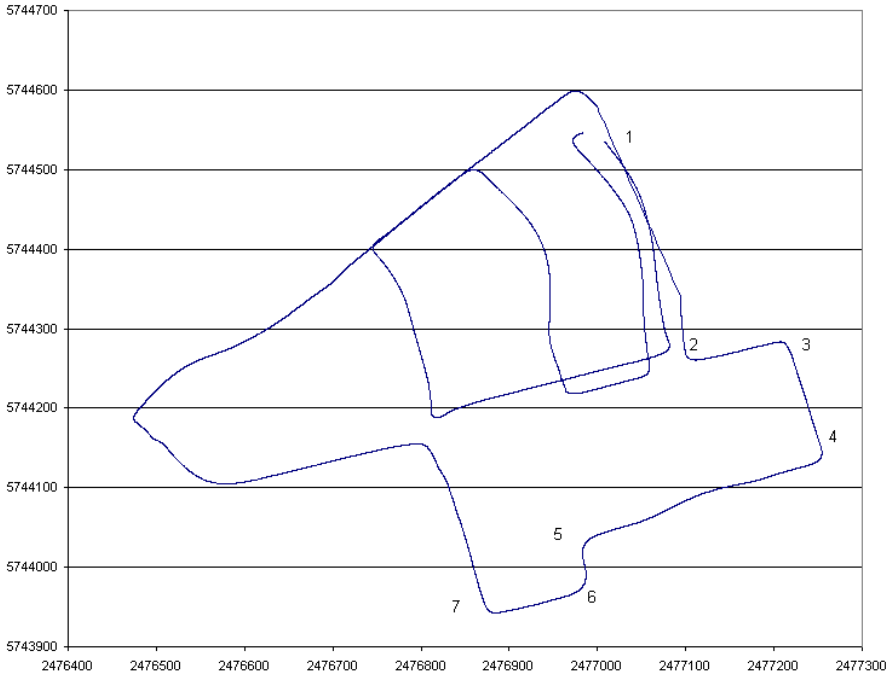

Figure 5.23. Selected data set presented as a series of positions in the New Zealand Map Grid coordinate system.

same streets, which is not a usual driver's behaviour. This was done to reduce the driver's strain, which is otherwise increased by the presence of very bulky equipment in the car. We believe that this fact does not affect the results of our experiments.

The test drive described above spans more than ten minutes of driving. During this time data from sensors are collected at 50ms intervals. During driving, the car was stationary for some periods of time on intersections, and there were a few long straight sections. As discussed previously, neural networks could easily "forget" previously learned material if they are exposed to different data values for long periods. To avoid such problems we manually edited longer stretches of similar values. As this affects only the learning capacity of the network, it should have no significant impact on the conclusions drawn from our experiments.

The data acquisition sub-system collects only "raw" outputs from analogue–to–digital converters (integer values as provided from a 12-bit output register). These values are transformed to voltage levels during post-processing. As described in Section 4.6. some digital processing was performed in order to improve the data and remove

noise. Firstly, systematic error was removed. For a number of different reasons, 0*g* levels for accelerometers are not exactly 2.5V, as designed. We averaged data from a large number of different test drives to calculate the median voltage levels for 0*g* output for both accelerometers. Calculated values are deducted from actual values, which provided 0V output for 0*g* acceleration. This step was necessary for easy application of the normalized low pass second order Butterworth filter (as described in Section 4.6.). Digital filtering was very effective and the resulting signal is a very accurate representation of actual vehicle accelerations.

Vehicle longitudinal acceleration is generally much stronger than lateral acceleration, as vehicle actuators (engine and brakes) mainly apply force in a longitudinal direction. As sensitivities for accelerometers are the same (theoretically) we receive different ranges for the input signals. If we apply such different signals to the neural network, the signal with larger numerical values will become dominant and will have more influence on the output from the network. This is obviously inappropriate as some important patterns in lateral acceleration are completely "obscured" by larger, but not so important, signals in longitudinal acceleration. In order to prevent this, there are a few possible options. Neural network weights could be preset to allow for different input ranges. This approach is easy, but it is not a very robust solution as weights can quickly change if the input signals we apply at the beginning are not representative. We selected another approach and normalized values from both accelerometers to the range [–1, +1]. Therefore, no sensor is favoured, and both training and testing are greatly simplified. Normalization parameters are calculated by analysing a large amount of collected data in various test drives.

Figure 5.24 displays manually edited, filtered and normalized values for longitudinal and lateral accelerations as applied to neural network inputs. Patterns representing left and right turns could be easily recognized, as each turn consists of one negative impulse followed by a series of positive impulses for longitudinal acceleration, while lateral acceleration has a negative or positive impulse depending on the type of turn (left or right). For easier understanding of driving patterns, we marked the first seven turns (the same ones we marked in Figure 5.23.) with numbers 1 to 7 on this diagram. Left turns (2 and 5) are marked above the diagram, as lateral acceleration has

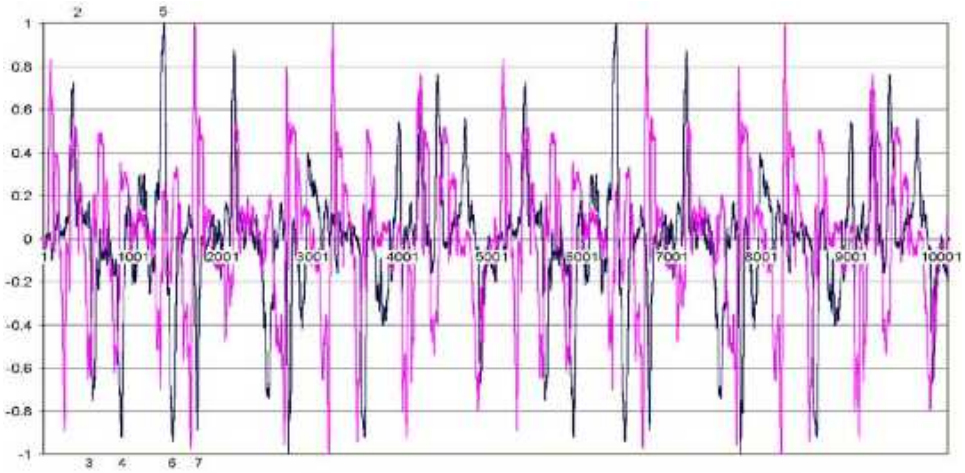

Figure 5.24. Longitudinal (pink) and lateral (blue) acceleration for selected data set.

positive values, while right turns (1, 3, 4, 7) are marked below the diagram, as lateral acceleration has negative values.

The resulting data set has 10,500 data samples. For our experiments, it is divided into three subsets:

- The *training set*, consisting of the first 7,300 data samples, which are used to train the neural networks;
- The *testing set*, consisting of the next 2,200 data samples, which are used to evaluate the prediction capabilities of neural networks, and
- The *cross-validation set*, consisting of the last 550 samples, which are used in the training phase to validate the learning process. It is evaluated as network is trained, and the training stops when error begins to rise.

5.4.3 Tools used in experiments

Data processing requirements for experiments in vehicle motion prediction by neural networks, as designed, are very high. A number of complex operations must be performed, such as: fast data collection, training and evaluation for various neural networks, and data visualization. Between these major operations, a number of smaller

operations such as data filtering, normalization and various other transformations must be performed.

We had very limited resources for our research, so we had to use the cheapest possible tools. As a result, our data processing tools represent a mixture of purpose built programs, "off-the-shelf" applications and, even, demonstration versions of software. Such on approach had a negative side in slowness and the need for additional data transformations (in order to make output from one program appropriate for input into another). However, we believe that the selected tools are fully appropriate for the experiments we conducted and there were no negative influences on the results

The software tools used for the experiments with neural networks are displayed in Figure 5.25. Data is transferred between tools using files. The data acquisition hardware and software are described in more detail in Chapter 4. The main goal of the data acquisition system is to enable reliable collection of data at predefined time intervals. In order to meet this goal and avoid possible problems, the data acquisition software collects data from analogue-to-digital converters (to which sensors are connected) and stores them in "raw" (unprocessed) form in a file. The data acquisition software also collects data from the GPS receiver, synchronizes them with sensor data and stores them to the same file. This approach minimizes the required processing power, and also minimizes problems which can arise from the non-real-time nature of the Windows operating system. In order to avoid manipulations with our mobile computer, the data acquisition software copies all collected data at the end of the session to a compressed file on a floppy disk. This enables easy data transfer to the Office PC and to other programs used in these experiments. The other important feature of the data acquisition software, also discussed in more detail in previous chapters, is its non-standard user interface, consisting of a very small subset of special keys and voice output.

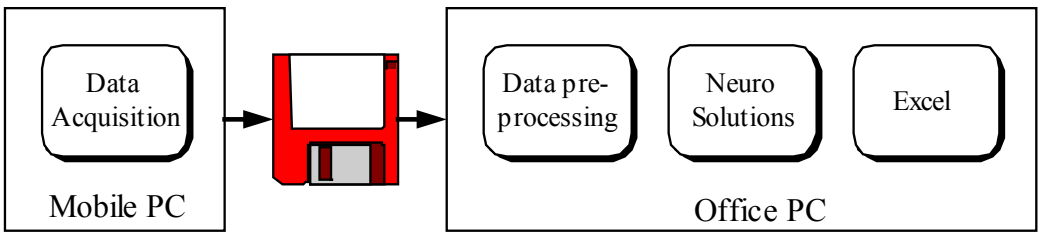

Figure 5.25. Software tools used in experiments.

The binary file created by the data acquisition software is not in a format convenient for use in the "off-the-shelf" software we used for the neural networks' training, testing and data visualization. In order to convert data to the appropriate formats, we developed a data pre-processing program. The main goals of this program are:

- To check the validity of data collected from the sensors and the GPS receiver.
- To extract data from sensors, convert values to voltages, change the 0*g* value to 0V, and perform digital filtration and normalization as described in the previous section. Converted data is converted to the format required by the NeuroSolutions software and stored in a text file.
- To extract data (positions) collected by the GPS receiver, convert coordinates to the New Zealand Map Grid coordinate system and store them in a text file.

The software we used for the neural network training and evaluation is *NeuroSolutions* produced by a company called NeuroDimension [Neuro, 2001]. NeuroSolutions is a neural network simulator that can be used for creating neural networks by using a wide variety of components: processing elements, connections, learning elements, etc. Such an approach enables experimentation with various neural network architectures, as described in previous sections. However, the user is not limited to "predefined" architectures, he/she can also mix components from various architectures according to needs. This approach in neural network simulation is marketed by NeuroDimensions as "Object-Oriented", as it resembles the object oriented approach in software development.

We used NeuroSolutions version 3.0, which was available at the time of the experiments. Due to limited resources, we used the free demonstration version available on their web site. Its simulation engine is the same as in the normal retail version, but most input/output functions are disabled. In order to overcome the limitations of the demonstration version, we had to develop and use a special software utility, which enabled saving training results.

More information about NeuroSolutions software can be found in the product documentation and an interactive book [Principe, 2000]. Here only the basic concepts necessary for understanding the discussion will be outlined. NeuroSolutions provides a large number of different *components* with which the user can build a neural network simulator. The workspace for network building is called a *breadboard*. Networks are constructed on the breadboard by selecting components from the palettes, stamping them on the breadboard, and then interconnecting them to form a network topology. Connections are made through the component's *access points*. Once the topology is established and its components have been configured, a simulation can be run. An example of a very simple one-layer perceptron is shown in Figure 5.26.

The most important NeuroSolutions components are:

- *Axon* components represent the set of processing elements in the network. Usually an Axon contains all neurons in a particular network layer. A number of different processing options are available in neurons, so there exists a large number of different types of Axons. For training purposes, Axons are complemented by BackAxon components which provide the necessary computation for back-propagation network training.
- *Synapse* components represent interconnections between Axons. There are different types of Synapses, which can provide full or partial connection between nodes in Axons. Appropriate BackSynapse components are used for network training.
- *GradientSearch* components complement BackAxon components in updating network weights. A BackAxon component propagates errors backwards through the

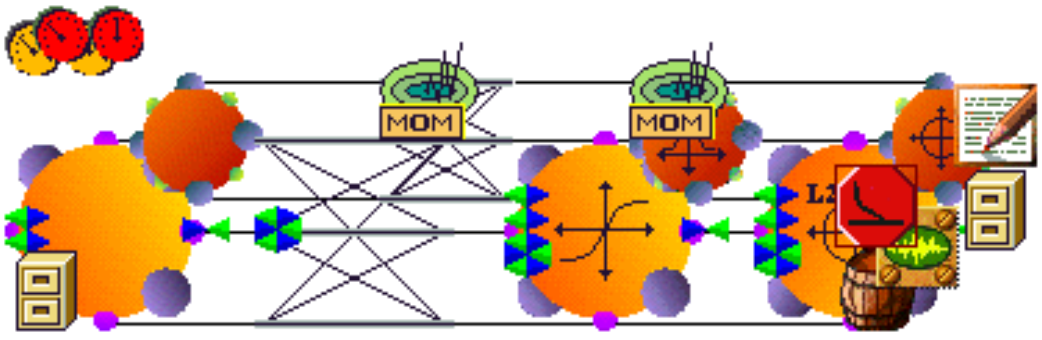

Figure 5.26. An example of a NeuroSolutions breadboard.

network and calculates weight gradients, as it knows the actual transfer function used in the Axon. GradientSearch components use weight gradients and apply them to calculate network weights for the next iteration.

- *ErrorCriteria* components calculate the network cost function. Supervised learning requires a metric, a measure of how the network is performing. ErrorCriteria components monitor the output of a network, compare it with some desired response and report any error to the appropriate learning procedure. In gradient descent learning, the metric is determined by calculating the sensitivity that a cost function has with respect to the network's output. This cost function is normally positive, but should decay towards zero as the network approaches the desired response. Several cost functions are commonly used, but the quadratic cost function (L2 in NeuroSolutions terminology) is by far the most widely applied.
- *File* represents an abstract object which supplies the network with data for training or evaluation. It reads data from operating system files (one or more), can provide translation into the appropriate format, and prepares the set of data for training or evaluation of the network. A few different data types are supported: textual, column–formatted, binary and bitmap files. There are also provisions for normalization and segmentation of input files.
- The *Control* component provides the user interface for setting and executing the network simulation. The user can select a data set (training or testing), load or save weights, set global simulation parameters, and start and stop the simulation.

NeuroSolutions supports a number of other components, notably a data visualization facility, which makes the system very suitable for educational purposes as it clearly displays the principles of network learning and evaluation. However, these components are not essential for a neural network simulation and will not be presented here. Figure 5.27 presents examples of the NeuroSolutions components described above.

We used Microsoft Excel for various ad–hoc data manipulations. Spreadsheet programs are very powerful in dealing with time series data. Excel provides many embedded statistical functions which can be used easily, and generates results quickly. For example, we pasted data received from NeuroSolutions as output of the neural network experiment, imported data from the file used as input for experiments, and,

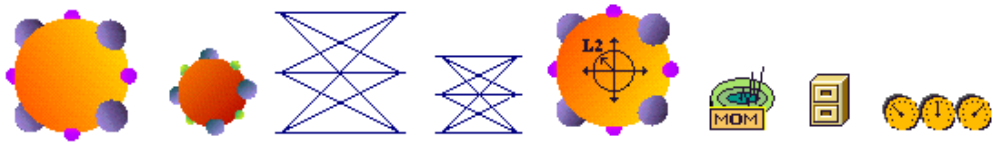

Figure 5.27. Examples of NeuroSolutions components: Axon, BackAxon, FullSynapse, BackFullSynapse, ErrorCritera, MomentumGradientSearch, File, and DynamicControl.

using a simple formula for prediction error (given in the next section), we calculated the total error for the executed experiment. Excel also has powerful visualization features, so we were able to quickly generate graphs of the input and output data (for a complete data set or for some parts of data) and easily confirm numerical results obtained. The majority of diagrams in this thesis have been generated in Excel. We also used the Excel programming model as a rapid prototyping environment for many algorithms. We developed macros for filtering, normalization and some other processing steps, and used Excel's methods for data loading and visualization for quick debugging and setup of various parameters. The developed algorithms were later re-coded in C++ as part of the data acquisition software, data pre-processing and other utilities.

5.4.4 Mathematical model of experiments

In the previous sections we presented data, processes and tools used for the experiments. In this section we will present the mathematical model used. The problem of predicting vehicle movement in terms of lateral and longitudinal acceleration is a relatively simple mathematical model defined by the following equations:

$$\hat{a}(t+n) = \varphi(a(t), b(t), a(t-1), b(t-1), \ldots, a(t-p), b(t-p))$$

$$\hat{b}(t+n) = \lambda(a(t), b(t), a(t-1), b(t-1), \ldots, a(t-p), b(t-p))$$

In these formulae, data values for lateral and longitudinal accelerations at the moment t are denoted by $a(t)$ and $b(t)$ respectively. *Predicted* values for lateral and longitudinal accelerations at the same moment are denoted by $\hat{a}(t)$ and $\hat{b}(t)$ respectively. The number of data epochs in the future for which we predict acceleration

values is denoted by *n*, the *prediction step*. The number of the previous data items which are used for prediction is denoted by *p*, the *prediction order*.

In our experiments, we decided to use two different values for the prediction step:

- A short term prediction for five time epochs in advance, which is equivalent to 250ms. The typical human reaction time is around 0.2s. We believe that if it is possible to detect a potential accident situation and warn the driver for 250ms in advance, this should provide enough time for the driver to react and avoid the accident. As such, a prediction step of 5 should be appropriate for low-level (reactive) navigation tasks.
- A longer term prediction for twenty time epochs in advance. A prediction step of 20 is equivalent to a duration of 1s. This interval could be enough for a driver to plan a more complex and conscious response to navigation problems.

The prediction order depends on the neural network architecture selected for a particular prediction task. Multi-layer perceptrons do not store any data from previous time epochs and their prediction order is equal to zero. Recurrent networks store data from previous time epochs in feedback connections and memory nodes. Their explicit prediction order is not easy to calculate, as previous values can influence predictions for many time epochs in the future. However, the strength of that influence decreases over time. Time-delay neural networks store explicit values from previous time epochs and their prediction order could be easily expressed as the number of tapped delay lines in each of the network processing nodes. In our experiments we used time-delay networks with 5 and 20 delay lines to match the numbers we specified for prediction steps.

The prediction performance of neural networks in our experiments were quantified by using a simple measure that we refer to as the *prediction error*. The prediction error is the sum of the squares of differences between predicted and observed (desired) values for all values in the testing data set. More formally, the prediction error can be expressed as:

$$\varepsilon = \sum_{t \in D_{TEST}} ((a(t) - \hat{a}(t))^2 + (b(t) - \hat{b}(t))^2)$$

In this formula, ε denotes the prediction error, and D_{TEST} denotes the test data set.

5.4.5 Determining prediction capabilities of a multi-layer perceptron

The goal of the first group of experiments was to determine the prediction capabilities of multi-layer perceptron networks. These experiments were conducted to prove that prediction capabilities of neural networks are appropriate for vehicle motion prediction, and that the results should serve as benchmark values for further experiments.

In these experiments we used a multi-layer perceptron neural network with three layers of neurons. The input layer consist of two nodes. On one of the input nodes we applied values for lateral acceleration; on the other input node we applied concurrent longitudinal acceleration. The hidden layer has four processing neurons, while the output layer has two neurons. Neurons in the hidden and output layers have a *sigmoid* transfer function. For easier adjustment of network parameters, the sigmoid function in NeuroSolutions is formulated as: $y(s) = \frac{1}{1+\beta \cdot e}$. Parameter β can be set up by the user – we selected a value of 1 to provide maximum sensitivity. A standard back-propagation learning algorithm with a momentum learning rule and a batch weight update was used.

NeuroSolutions' representation of the network used in these experiments is shown in Figure 5.28. Input, hidden and output Axons (layers) are denoted by numbers 1, 2 and 3 respectively. Hidden and output neurons have a sigmoid transfer function and they are represented by *SigmoidAxon* components. The transfer function has been displayed as part of the symbol. The input Axon has been connected to the File component. It represents a file from which lateral and longitudinal acceleration is read and fed into the input Axon. Axons are connected by *FullSynapse* components. Appropriate back-propagation components are used to enable network training: *BackAxon*, *BackSigmoidAxon* and *BackFullSynapse*. The gradient search is performed by the *Momentum* component which is denoted by number 6.

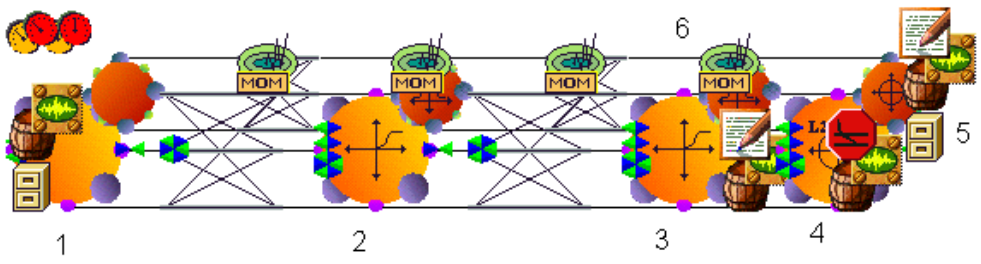

Figure 5.28. The multi-layer perceptron used in experiments.

Component 4 is an *L2Criterion* object. It calculates the quadratic network cost function using the formula: $J(t) = \frac{1}{2}\sum_i (d_i(t) - y_i(t))^2$. This component has a *BackCriteraComponent* attached to it, which communicates with other back-propagation components. File component 5 represents a file from which the network reads the desired values. As our goal is to predict, desired values are read from the same physical file as the input values, with an offset equal to the prediction step. The prediction step is set by the user when the network is created.

The network has a number of *DataStorage* and *Probe* components (*MegaScope* and *DataWriter*). We used a *DataWriter* component which is connected to the output Axon to display the predicted values generated by the output neurons. We used the DataWriter window and utility described in the previous section to save predicted data to the file.

The first experiment conducted with MLP was for a prediction step of one – we predicted what would be the acceleration 50 ms in advance. As expected, due to the mechanical nature of the vehicle's motion, accelerations did not change very much over this duration and network prediction was very accurate. We used this experiment to familiarize ourselves with the simulation environment and to set-up our experimental procedure. To check if the simulator worked as expected, we used only those parts of the training set which corresponded to right turns in our next experiment. Then we trained the network with that data set and re-evaluated the network with the complete test set. The resultant network was not able to correctly predict the lateral accelerations that are associated with left turns.

As expected, there were variations in the total error calculated after each testing session, as the network finds different local minimums. In order to reduce the influence of this effect, we decided to repeat each experiment several times and to take the average value for the total error as the final result of the experiment. The number of repeats is determined by variations in the errors. If error values are very similar, we do not need to repeat the experiment.

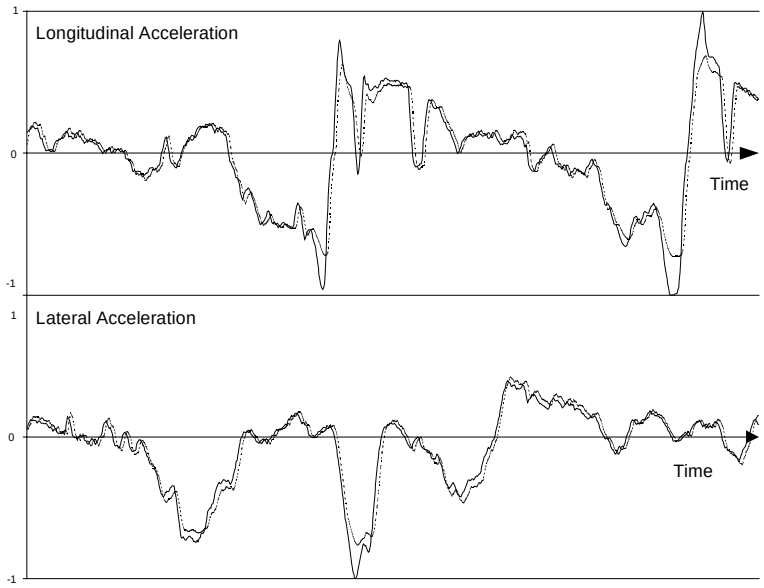

Figure 5.29. Predicted and actual values for lateral and longitudinal acceleration (250ms in advance).

For the next experiment we increased the prediction order to 5, which is equivalent to a period of 250ms. This period is large enough to provide for significant changes in acceleration. The numerical value for the total error we obtained was 6.90. However, the quality of the actual prediction could be judged by visually comparing the desired and predicted values as shown in Figure 5.29. It is clearly visible that the predicted values (presented as a dotted line) for both lateral and longitudinal accelerations are very close to the desired values – all important waveform features are preserved, except for the most extreme values. As described above, to calculate the total error we repeated the training and testing stages on the same data (with randomised weights between iterations as in all experiments). The results of repeated experiments were very similar, which suggests that the prediction performances for MLP and 250 ms in advance are very good and robust.

In the next stage we experimented with prediction for longer periods and increased the prediction step to 10 and 20. Respective times are 500 ms (0.5 s) and 1000 ms (1 second). These values were selected, because drivers could make more conscious and higher-level decisions during such an interval. The total error received as a result of the neural network simulation was significantly higher: 26.22 and 58.66. The predicted

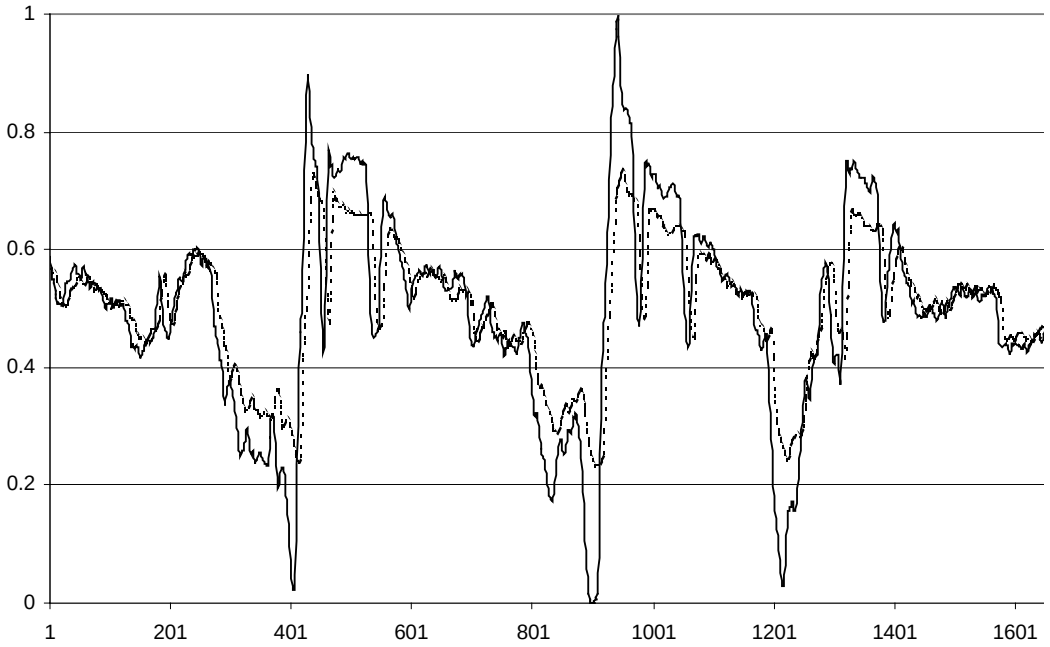

Figure 5.30. Predicted (dotted line) and actual values (solid line)
for longitudinal acceleration (1s in advance).

acceleration values became different from the actual values in some areas, particularly in periods when accelerations had larger values, as can been seen in Figure 5.30. It is easy to conclude that the predicted data during braking periods are lost in almost all cases. Unfortunately, these periods are of special interest for us as they represent turns and other significant driving events when support for the driver is required more than in normal circumstances.

The other important finding was that neural networks with longer prediction periods required a much larger number of training iterations. When we set the prediction step to 20 we required approximately 30,000 training iterations. This is much larger compared to the 10,000 training iterations that were required when the prediction step was 5. As computational power is still limited it was obvious that the number of available training iterations was a significant parameter which should be investigated further, as was done in the following experiments.

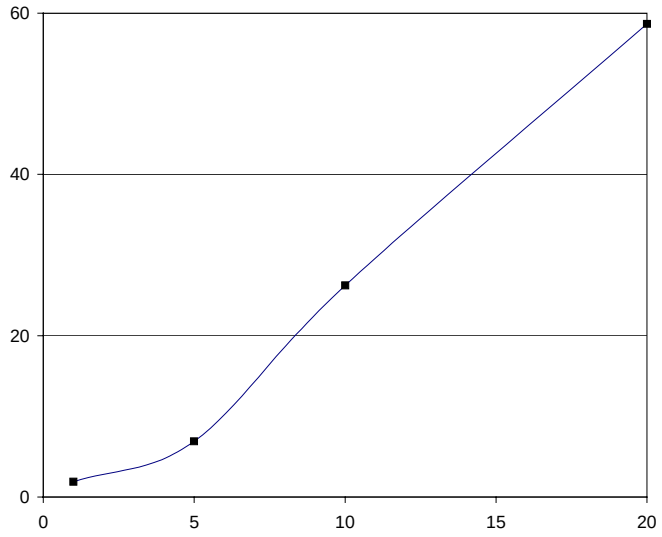

Figure 5.31. Total prediction error for multi-layer perceptron
for different prediction steps:
(1: 1.88; 5: 6.91; 10: 26.23; 20: 58.66)

A summary of the results of the experiments with MLP is shown in Figure 5.31. It can be seen that the total prediction error is small and slowly increases for small prediction order values. However, as the prediction order increases, the total error becomes larger and increases much faster. Predicted vehicle motion for shorter periods of time could be used to support drivers in executing reactive navigation tasks. However, we believe that longer term predictions by a multi-layer perceptron are not useful for vehicle navigation.

To provide a baseline for assessing neural network performances, we introduce the simplest statistical forecasting technique: the Moving Average. We compare prediction capabilities of the MLP and Moving Average (MA) algorithms for one second in advance. The Moving Average algorithm uses the current and 9 previous values for accelerations. Figure 5.32 illustrates the actual and predicted values (by both MLP and MA) for the test data set presented in Figure 5.30. In this graph we display only a portion of original data to make differences between predicted values more obvious.

As discussed in Section 5.2, the MA and similar techniques are not capable of dealing with non-linear and highly dynamic data, as is the case with the vehicle acceleration data. This could be easily seen from the results of this experiment. The MA

Figure 5.32. Prediction capabilities of multi-layer perceptron compared with the Moving Average method.

predictions follow the original data with filtered-out high frequency components and shifted in time for 20 epochs (1 second). MLP predictions may have problems predicting extreme acceleration values, but are much more accurate then MA predictions in most situations. The total prediction error for MA was 557.50 compared with MLP error of 58.66.

5.4.6 Comparing Prediction Capabilities of a Multi-layer Perceptron with More Complex Architectures

The goal of the second set of experiments was to compare the prediction capabilities of multi–layer perceptrons with those of neural network architectures recommended for time–series predictions: time-delay neural networks and recurrent networks. We showed previously that multi–layer perceptrons can predict vehicle acceleration well for short periods in advance. However, their longer term predictions are much worse. Here we would like to explore whether significant gains can be achieved by using more complex architectures.

The architecture of the multi–layer perceptron network we used in these experiments is exactly the same as the one used for previous experiments. It is a two–layer network with four hidden nodes and a sigmoid transfer function.

Time–delay neural networks (TDNN) are described in more detail in Section 5.2.2. They represent an extension of multi–layer perceptrons with delay lines that store the input data history for each processing node. We used TDNNs with two different numbers of delay elements (5 and 20) connecting the input and hidden network layers. The number of delay elements between the hidden and output layers is 2 and 10 respectively. As a result, we have an equivalent network of 10 or 40 input nodes, and 8 or 40 hidden nodes, while the number of output nodes always remains 2. As can be seen in Figure 5.33, most of the network parameters are the same as for the multi–layer perceptron: a sigmoid transfer function, a momentum gradient search and a quadratic network cost function.

Reccurent neural networks are also described in Section 5.2.2. They represent a large class of various network architectures having recurrent feedback connections as a common feature. The architecture of the recurrent network used in our experiments is shown in Figure 5.34. It is designed to be as similar as possible to MLP and TDNN.

The hidden layer of the network consists of six neurons, which are fully connected to themselves through delay elements with a delay of one time epoch (presented in Figure 5.33 as z^{-1}). Connections between the input and hidden layer contain arrays of six

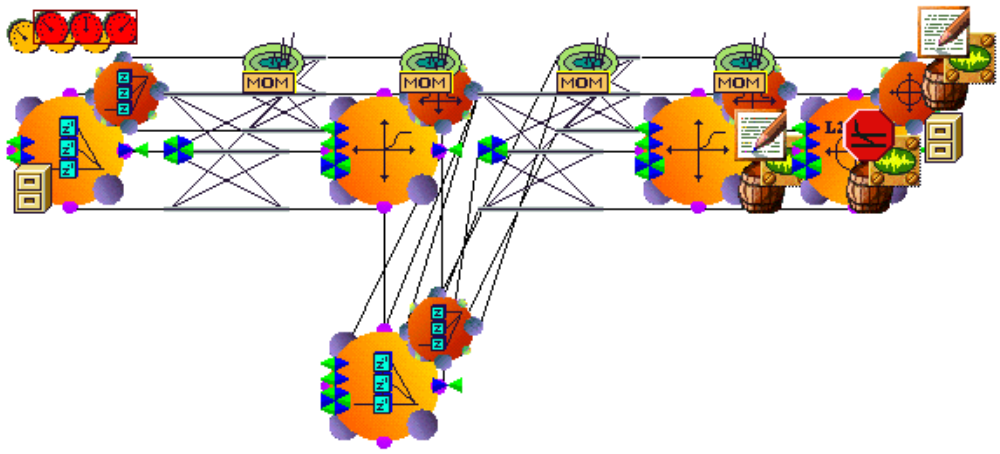

Figure 5.33. Time–delay neural network used in experiments

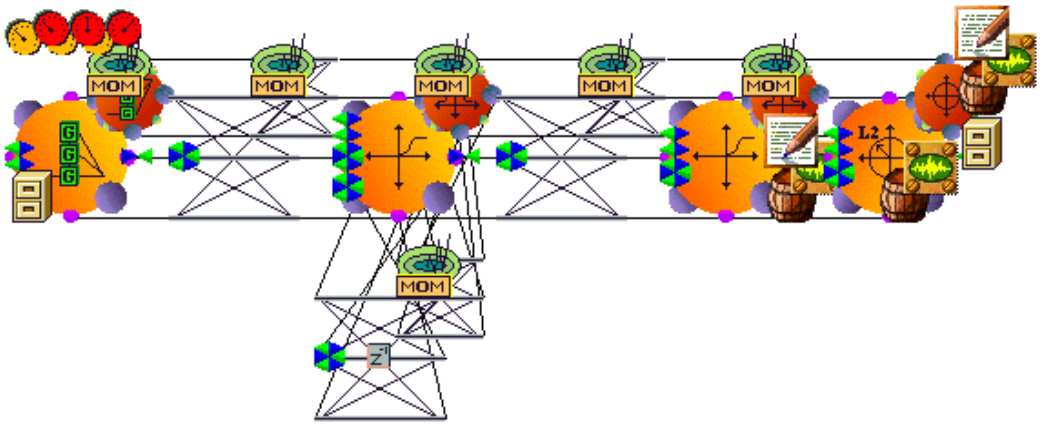

Figure 5.34. Recurrent neural network used in experiments.

delay elements with a gamma kernel function (as described in Figure 5.14.). Gamma delay elements provide memorize the input signal history and behave as leaky integrators (as shown in Figure 5.14.c).

In these experiments we investigated the prediction capabilities for two different values of prediction step: 5 and 20. We already found that a multi–layer perceptron's prediction capabilities for five time epochs in advance are very good. For a prediction step of twenty, we found that MLP's prediction capabilities are not appropriate as it fails to generate correct predictions for most of the acceleration values. With this experiment we wanted to discover whether more complex architectures could provide useful predictions over a longer time period.

As noted in the previous section, results also depend on the maximum number of training iterations allowed. This becomes more important if we predict further into the future. When we use more complex architectures the number of required training iterations could become very large. This is important, since a large number of training iterations could make the online training required for driving support intractable.

For the first group of experiments we set the prediction step to n = 5. We used four different architectures (multi–layer perceptron, recurrent network and two variations of time–delay networks) and five different values for the maximum number of training iterations. The results of this group of experiments are presented in Figure 5.35 and Table 4.1.

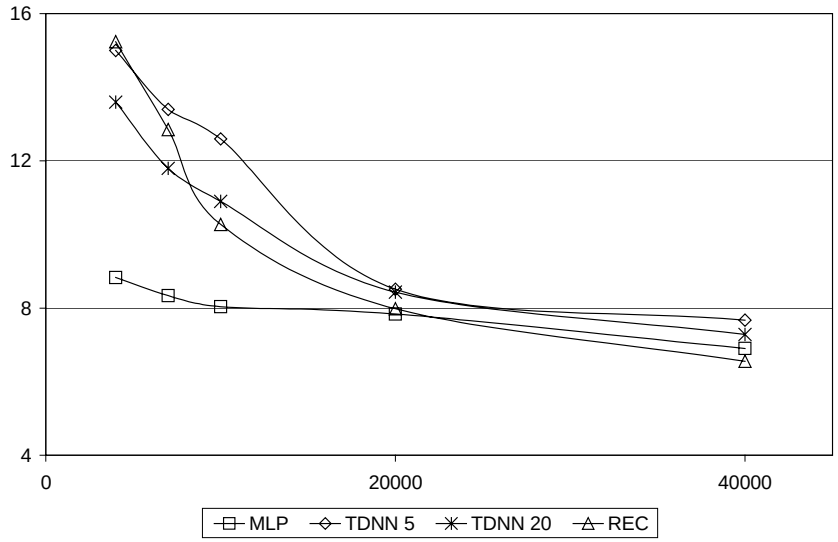

Figure 5.35. Total prediction error for prediction step $n = 5$.

	MLP	TDNN 5	TDNN 20	REC
4000	8.83	15	13.6	15.24
7000	8.34	13.4	11.8	12.85
10000	8.04	12.6	10.9	10.27
20000	7.84	8.51	8.43	7.98
40000	6.9	7.67	7.28	6.55

Table 5.1. Total prediction error for prediction step $n = 5$.

As found in previous experiments, the prediction capabilities of the multi–layer perceptron for $n = 5$ are very good. It is also expected that MLP, as the simplest architecture, requires fewer training iterations than more complex architectures, and that its results do not improve significantly if training is extended. As we could also expect, a TDNN with a larger number of delay elements (20) generates better predictions than a TDNN with only five delay elements. However, it was surprising to discover that MLP performs better than TDNN even after more training iterations, achieving the second best result. We believe that this is caused by the relatively simple nature of the problem, which enables the multi–layer perceptron to work well enough for it to be difficult for complex architectures to produce better results. From this we can conclude that the

prediction capabilities for short periods in advance are good for all tested network architectures. However, since it is obvious that the simplest of them, a multi–layer perceptron, has sufficiently good prediction performances, it seems not justified to use architectures that are far more complex.

In the second group of experiments we tested the longer term predictions of various neural network architectures. Here we set the prediction step to $n = 20$. We previously determined that the prediction capabilities of a multi–layer perceptron in this case were not sufficiently accurate. We also noticed in previous experiments that a TDNN with twenty delay elements achieves better results than the five element version, so we decided not to include the latter in our experiments.

From the first simulations we deducted that the number of required training iterations would be much higher than for shorter–term predictions, so we used two limits: 40,000 and 80,000. The training of complex architectures (TDNN and recurrent), even for the smaller number of iterations, is very time consuming. We believe that with careful selection of training data this time could be reduced. The training buffer keeping a representative set of input data, similar to one used in the system ALVINN (described in Section 5.3) can be applied.

The results for longer term predictions are presented in Figure 5.36 and Table 4.2. It is obvious that the total prediction error for all architectures is much higher than for the smaller prediction steps. Neural network architectures which are recommended for time–series prediction also performed significantly better than multi–layer perceptrons. However, the final prediction error is still high. Even when we allowed a very large number of training iterations, results are not sufficiently improved to warrant the usability of predicted data for driver support.

Number of training iterations	MLP	TDNN	RECC
40000	58.66	41.8	48.14
80000	55.45	41.3	44.1

Table 5.2. Total prediction error for prediction step $n = 20$.

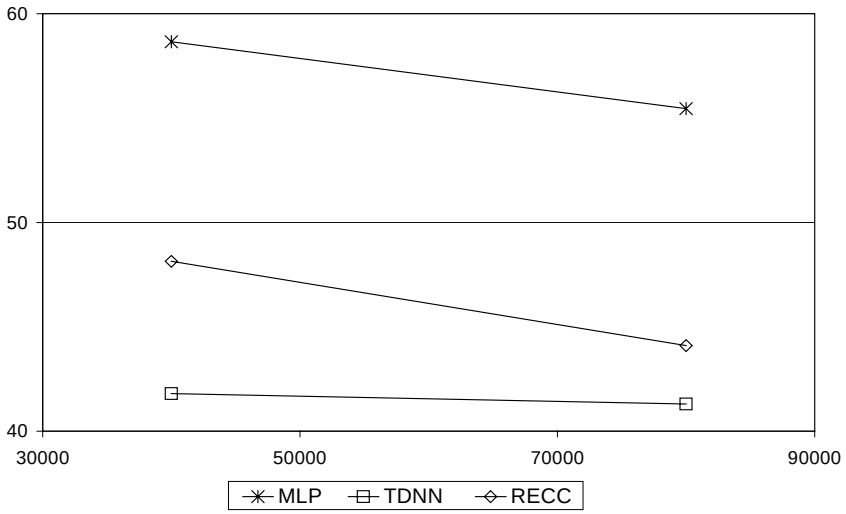

Figure 5.36. Total prediction error for prediction step $n = 20$.

It is likely that fine-tuning of the network architecture and parameters could improve prediction performance. It may also be possible to develop a more sophisticated approach in selecting training data to further improve network performance. However, we believe that any network which could be implemented for on–line training and prediction, cannot achieve the longer–term prediction capabilities required for an effective driver support system.

5.4.7 Conclusion

The results of our experiments in short-term vehicle motion prediction by neural networks were presented in this Chapter. Neural networks are a computation model developed to mimic the human brain's information processing architecture. In this chapter we presented basic neural network concepts as well as a short history of neural network research. Special attention was paid to neural network training by back–propagation learning, which is still the most widely used training technique. Neural networks are well known for their time–series prediction capabilities. A number of important neural network applications for forecasting purposes were discussed and we also presented the most important architectures used for time–series prediction.

Neural networks are used in intelligent transportation systems to fulfil a number of different roles, including the predictions of various parameters. We presented a selection of research achievements. The most popular is the ALVINN system, developed at the Carnegie Mellon University Robotic Institute. ALVINN is a system for generating steering commands based on incoming images from a video camera and/or laser range-finder. The authors claim that a vehicle equipped with ALVINN successfully drove large distances.

We conducted experiments to prove that short-term vehicle movement prediction is possible using a very limited set of sensors and well-known machine learning techniques. The prediction performances of the neural networks for time-series data from vehicle sensors were analysed and the prediction capabilities of various neural architectures in this domain were compared.

In our experiments we used NeuroSolutions' neural network simulator and a number of other utilities. We selected a data set from a large number of available test drives according to the described criteria. This data set had been edited and pre-processed to make experiments with neural networks easier. It had also been split into training, testing and cross–validation subsets.

The first set of experiments was based on a multi–layer perceptron. The results suggested that the prediction capabilities for small values of prediction order are very good and that predicted accelerations are very close to actual ones. However, it also become obvious that prediction error increases substantially for larger prediction orders. In these circumstances, networks are not able to correctly predict accelerations for driving events such as turns. However, such events are of major interest for any effective driving support system.

In the second set of experiments we compared the prediction capabilities for various neural network architectures for a short period of time in advance. We found that there is no major difference in results among architectures and that prediction capabilities are reasonably good. Multi-layer perceptrons performed comparably with more complex architectures and were much quicker in training.

The third set of experiments explored longer–term predictions with various architectures. This time, time–delay and recurrent networks performed better than

multi–layer perceptrons. However, predicted accelerations were still very different from actual values and the required processing power for training becomes very high. We may conclude that longer term prediction by neural networks is not easy to achieve and that symbolic machine learning techniques should be used instead.

6. Driving Event Recognition and Long Term Prediction

Goals:

In this Chapter we present the results of our experiments in long-term symbolic prediction of driving events. We developed a reliable method for driving event recognition by hidden Markov models. Hidden Markov models (HMM) are often used to recognize patterns in time-series data. We found that they could be very successful in recognizing driving events from data from a very limited set of sensors. We developed program WinHMM for training and evaluation of hidden Markov models.

Recognized driving events are then used to predict the future behaviour of the driver and the vehicle. We developed a prediction methodology based on a simple data structure for storing previous experience and the concurrent evaluation of multiple hypotheses. The results of our experiments proved that our system is able to reliably predict future driving events, based on a small set of previous experiences.

Results described in this chapter are also presented in the following papers:

Mitrovic D., Driving event recognition by Hidden Markov Models, in *Proceedings of the International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting services*, pp. 110-113. 1999.

Mitrovic D., Reliable Method for Driving Events Recognition, accepted for publication in *IEEE Transactions of Intelligent Transportation Systems*.

6.1. Introduction

One of the conclusions of our experiments with neural networks is that they are good in predicting vehicle motion for a short time in advance. However, their long-term prediction performance is not as good. We believe that symbolic pattern learning techniques must be used in order to provide reliable prediction of future driving events.

Learning of temporal sequences is used in numerous applications. Speech recognition is a representative example where research started decades ago and led to successful commercial solutions. Today even not very powerful computing platforms, such as mobile phones, can recognize spoken commands. The Reactive Keyboard is a computer program developed to increase the speed and ease of communicating with a computer, especially for physically disabled people [Darragh, 1990]. It predicts the sequences of characters that the user would like to type, based on the stored, previously typed sequences and the current sequence of characters. It uses relatively simple methods to achieve a significant level of help for some users. In this sense it provides inspiration for the research presented here.

For our research we will develop the following model. Someone's driving consists of driving events. We use the term *Driving event* to refer to any major change in vehicle position, attitude or speed, and will denote it as a_i. We may theoretically define many different types of driving events; however, only a limited number of types of driving events are a part of the common wisdom, such as: left and right turns, driving along the straight road, etc. We will denote the type of the particular event with T_{ai}. The type characterizes driving events, but two driving events of the same type could be very different. For example, two road segments could be both straight, but may have very different lengths or different speed limits. Because of that, driving events must be also characterized by other relevant parameters like duration of the event, the average and maximum speed, the average and maximum acceleration (longitudinal and lateral), speed and acceleration variances, and so on.

Places where vehicles are stationary for a long period of time are of special interest. Recent research [Ashbrook, 2002] showed that there is only a limited number of such places and their number decreases as we increase the time threshold (the required

duration of the vehicle being stationary). We will denote such stationary places as S_i. The ordered set of driving events, while driving from place A to place B, is called a *route*. Mathematically, the route *R* can be defined as an ordered set of driving events:

$$R: S_A \rightarrow S_B = \{a_1, a_2, \ldots, a_n\}.$$

The goal of our research is to predict the future driving events based on the current vehicle state and the previously experienced sequences of driving events. The solution for this problem consists of the following steps:

1. The current driving event type must be recognized, based on data received from sensors.
2. This event must be further described by other relevant data collected by the sensors.
3. The current event must be compared with similar events from previously experienced routes.
4. One or more hypotheses should be generated and evaluated.
5. The future event is predicted from hypotheses with the largest similarity to the currently observed set of events.

In this chapter we will present methodologies and tools we developed to predict future events from learned driving patterns. We will present our experiments and their results, which support our claim that the developed technique can successfully predict future events. We will start with the driving event recognition by using hidden Markov models. In the next section a brief introduction to the hidden Markov models is therefore given.

6.2. Hidden Markov Models

6.2.1 History

The question whether the character of nature is deterministic or probabilistic, is as old as humankind. It is treated as a philosophical question, since no simple scientific answer can be found. As it was much simpler and easier to understand by ordinary people the deterministic view usually prevailed during the history. Probabilistic concepts were hard

to understand even for mathematicians, and the theory of probability was developed relatively late, by Bayes and Gauss in the 18th and 19th centuries. However, a number of scientific inventions during the 19th and 20th century, such as Brown's particle motion and Einstein's general theory of relativity, brought more evidence that the character of the nature is in fact probabilistic.

In the period between the two world wars, and particularly during the second world war, research into stochastic processes was greatly intensified, as many new scientific discoveries could not be explained by deterministic methods. The advent of electronic computers marked the era of intense use of stochastic models to solve various problems. During the 1950s statisticians tried to solve the problem of characterizing random processes, for which only incomplete observations were available [Hartley, 1958]. His approach was to model the original system as a random process and the procedure of generating observations as another random process. Then both the original and the observation generation processes are characterized from observed data. This concept of "doubly stochastic process" is now known as a *hidden Markov model* (HMM).

During the sixties and early seventies, Baum and his colleagues published the basic theory of hidden Markov models in a series of papers [Baum 1966-1972]. The major result of these papers was the Baum-Welch re-estimation algorithm. The Baum-Welch algorithm made it possible to calculate hidden Markov model parameters from a given set of observation sequences in time, which is linearly proportional to the length of the observation sequence. It enabled the practical implementation of hidden Markov models on electronic computers, since this problem was computationally intractable otherwise.

Baum's papers proposed an abstract solution, which was only marginally accepted by the engineering community. During the seventies, HMMs have been implemented in speech recognition applications at Carnegie Melon University [Baker, 1975] and by a research group lead by Jelinek at IBM [Jelinek, 1975]. However, first published results about systems for speech recognition influenced a number of other researchers to start using HMM.

During the eighties and early nineties a few very detailed tutorials about how to use HMM for speech processing were published [Rabiner, 1989][Deller, 1993] and

paved the way for much wider use of HMM. During the same time many other statistical techniques such as Bayesian networks, etc have become popular. Computing hardware also progressed significantly, making computationally intense HMM training possible, even on small personal computers. The nineties witnessed the use of hidden Markov models in a very wide range of applications, from signature verification to human genome investigations. Hidden Markov models enabled reliable recognition of temporal sequences in many different domains, which is why we decided to use them to recognize driving events.

The problem domain could be continuous or discrete and hidden Markov models for solving the problem could be therefore continuous or discrete. In the next chapter, the theoretical foundation of discrete hidden Markov models is given. Continuous hidden Markov models follow the same theoretical foundation, with slightly different formulae.

6.2.2 Theoretical Foundations

The foundation for discrete hidden Markov models is the concept of a discrete Markov process. A discrete Markov process is a stochastic process which consists of a discrete random variable, finite space state and a discrete-time index. An important characteristic of Markov processes is the Markov property, defined by the following formula:

$$P(X_{t+1} = j \mid X_t = i, X_{t-1} = i_{t-1}, X_{t-2} = i_{t-2}, \cdots, X_0 = i_0) = P(X_{t+1} = j \mid X_t = i)$$

For general stochastic processes, the probability of transition from state *0* to state *t+1* depends on all previous states in which the process were in (i_0, i_1, ⋯,i_{t+1}). The markov property reduces dependancy only to the current process state *t.* Given the current state, any other information about the past is irrelevant for predicting the next state. Markov processes are very important, as they are computationaly feasible and it has been shown that many stochastic processes have Markov property, or could be conviniently presented as Markovian.

A discrete Markov process has a finite number of states, usually not very large. At discrete time intervals it goes from one state to another (or to the same state), according to a set of transition probabilities. The simplest example of a Markov process is a frog in a lily pond with a number of lily pads. The frog sits on one of the lily pads and

occasionally jumps, either to the same pad or to another. If the frog is not influenced by some external stimulus, jumps are random and do not depend on the history of previous jumps. Probabilities for jumping from any pad to another are formalized in the state transition matrix for the model.

In some cases, state changes in the underlying Markov process are hidden from the user (i.e. they are not observable). However, the user can observe process changes through the sequence of observations which are produced at each change of state as shown in Figure 6.1. Due to a number of reasons (observation measurements error and problems, for example), the observation process could be characterized as another stochastic process.

To illustrate basic HMM concepts a simple but informative "urn and ball" model has been used in literature (introduced by Jack Ferguson and published in [Rabiner, 1989]). We assume that N large urns are in a room and in each urn there are a large number of coloured balls. There are M distinct ball colours. Someone randomly chooses an urn and then, again randomly, chooses a ball from the selected urn. The colour of the selected ball is recorded as an observation and the ball is returned to the urn. The process continues by selecting the next urn. It is easy to conclude that there are two separate stochastic processes, one for urn selection and another for ball selection. In both cases, the Markov property is valid and we obtain a finite observation sequence of colours as the output of the process.

Observations generated by a hidden Markov model could be continuous signals or discrete symbols from the finite alphabet. The stochastic process of observation

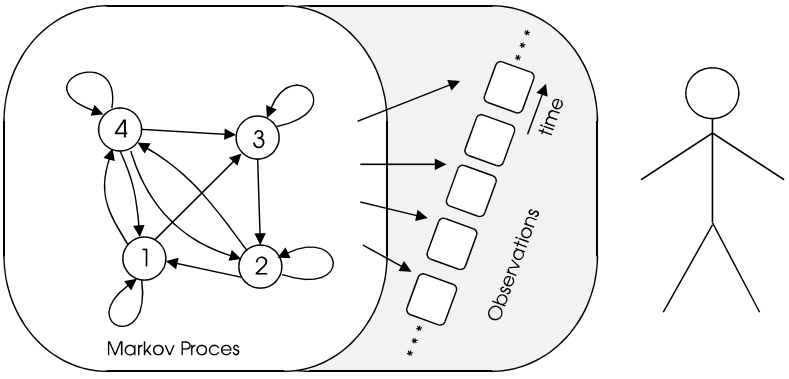

Figure 6.1. The basic concepts of hidden Markov models.

generation is defined by an output probability distributions for each state. A probability distribution can be discrete or continuous, depending on the type of observations. Both types have their own advantages, but the low computation cost of discrete models made them much more popular.

More formally, discrete HMM can be characterized by:

- A set of N distinct states $S = \{S_1, S_2, \ldots, S_N\}$, with q_t denoting a state at time t. States are hidden and in some applications they do not have physical significance. However, in some applications a state could be easily associated with physical objects or events. The "urn and ball" model has one state for each urn.

  - The initial state distribution $\pi = \{\pi_i\}$ where $\pi_i = P[q_1 = S_i] \quad 1 \le i \le N$.

  - The state transition probability distribution $A = \{a_{ij}\}$, where
    $$a_{ij} = P\left[q_{t+1} = S_j \middle| q_t = S_i\right] \quad 1 \le i, j \le N .$$
    It is obvious that by definition $\sum_{i=1}^{N} a_{ij} = 1$.

  - Each state can produce one of M distinct observation symbols from the set $V = \{V_1, V_2, \ldots, V_M\}$.

  - The observation symbol probability distribution in state j, $B = \{b_j(k)\}$, where
    $$b_j(k) = P\left[v_k \text{ at } t \middle| q_t = S_j\right] \quad 1 \le j \le N \quad 1 \le k \le M .$$

  - Again, by definition we have $\sum_{j=1}^{N} b_j(k) = 1$, and this should be preserved when the probability distribution is calculated.

Therefore, a hidden Markov model λ can be specified as $\lambda = (N, M, A, B, \pi)$ or, as N and M are implicitly defined in A and B, more compactly as

$$\lambda = (A, B, \pi).$$

The probability that observation sequence $O = O_1 O_2 \cdots O_T$ is generated by the model λ is denoted as $P(O|\lambda)$. Each observation O_t is a symbol from the set V, and T is the number of observations in the sequence.

There could be a number of different HMM architectures, depending on the limitations imposed on the state transition probability matrix *A*. If we impose constraints $a_{ij} = 0 \quad \text{for} \, j < i$ and $\pi_1 = 1$, we get a so called *Left-to-right model*. A Left-to-right model always starts at the first ("leftmost") state and transitions are allowed only toward later ("right") states or to the same state. It has been shown that the left-to-right model captures dynamic characteristics of data better than the general model, by imposing a temporal order on the model [Yang, 1995]. Left-to-right models are used in a number of temporal pattern recognition applications like speech recognition, human gesture recognition, and in other areas with great success. As the driving event recognition problem is similar to the above mentioned domains we restricted our HMM investigations to Left-to-right models. Various HMM architectures are presented in Figure 6.2.

Two basic operations for HMM pattern recognition are training and evaluation. *Training* is a procedure of calculating the model parameters (namely, state transition and observation symbol probability distribution and initial state distribution for general HMMs) on the basis of a training set of observation sequences, in order to maximize *P(O|λ)*. Generally, the topology of the model (number of states, number of symbols and imposed constraints) is known before training starts. We should also note that the initial state distribution for left-to-right models is fixed and does not need to be recalculated.

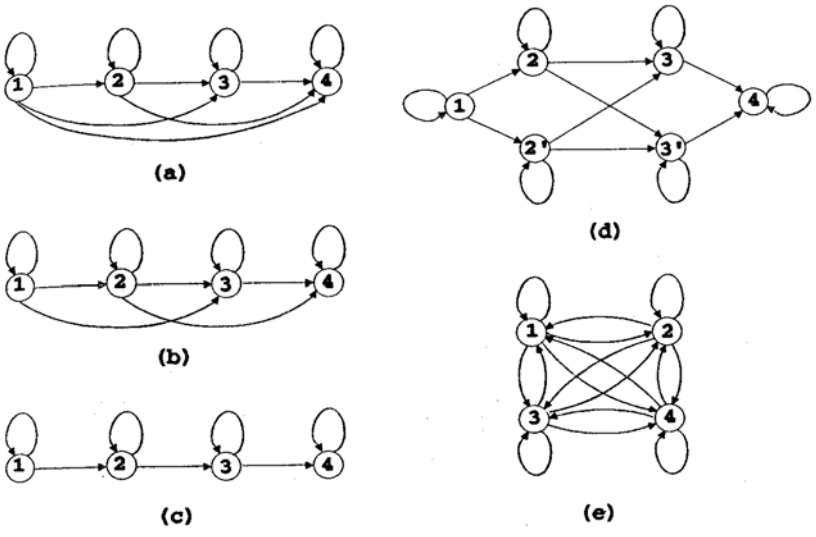

Figure 6.2. Various HMM architectures.

Evaluation is a procedure of calculating the probability of a particular observation sequence being generated by the model with given parameters ($P(O|\lambda)$).

There are two additional operations for HMMs: finding the optimal state sequence and generation of observation sequences. The optimal state sequence is a set of states which are most likely to produce a given observation string for a given model. The procedure for finding the optimal state sequence is usually closely related to the evaluation procedure. Observation sequence generation for a given model is a straightforward task. Numbers in the range [0-1) are generated by a random number generator and then applied to transition and output probability matrices to determine the next model state and the next output symbol. The observation sequence is finished when the model reaches finishing state, or the required sequence length is obtained.

6.2.3 Algorithms for Training and Evaluation

For training, we used the popular Baum-Welch re-estimation method. This is an iterative procedure which starts with initial model parameters that are set randomly or uniformly (or by using the appropriate procedure). Training sequences are evaluated using the *forward-backward procedure*. The forward variable $\alpha_t(i)$ represents the probability that an observation sequence $O_1, O_2,\ldots,O_t$ is generated until time t, and that the state at time t is S_i, for the given model: $\alpha_t(i) = P(O_1, O_2, \cdots, O_t, q_t = S_i \mid \lambda)$.

In a similar manner, a backward variable $\beta_t(i)$ is defined as the probability that the model is in state S_i at the time t, and that the future observation sequence will be $O_{t+1}, O_{t+2},\ldots,O_{t+N}$. Both forward and backward variables are calculated through induction, starting from the first or last time epoch. In the Baum-Welch procedure, the calculated forward and backward variables are iteratively used to re-estimate model parameters using the following formulas:

$$\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i) \beta_t(i)}, \qquad \overline{b}_{jk} = \frac{\sum_{t=1}^{T-1} \alpha_t(j) \beta_t(j)}{\sum_{t=1}^{T-1} \alpha_t(j) \beta_t(j)} \; .$$

Evaluation and re-estimation steps are repeated until we reach the local probability maxima for $P(O|\lambda)$. In order to find a better global solution, the Baum-Welch procedure is usually repeated several times. Those parameters (*A* and *B)* of the iteration which generated the highest probability are selected as the training result.

The evaluation of the observation sequence on the trained model is a much simpler procedure. It may be conducted by using the described forward-backward method, as the forward variable calculated for a complete observation sequence and the required finishing state actually represents the probability that a sequence is generated by the given model. The backward variable is not required for evaluation. The *Viterbi Algorithm* [Forney, 1973] is another very popular method for evaluation of observation sequences. It is very similar to the forward calculation, except its implementation is more efficient. It is also possible to organize the training of the hidden Markov models by using the Viterbi algorithm, but this requires a larger number of training sequences to generate good model parameters.

6.3. Hidden Markov Models Applications

In this section we review a few applications of hidden Markov models. Some of these applications are not directly related to transportation research, but illustrate well the time-series recognition capabilities of HMM. They demonstrate methods for the processing of data for use in HMM, selection of the HMM architecture based on domain specific knowledge, and approaches for evaluation of HMM prediction performance. In this sense, presented applications represent a background we used while developing the driving event recognition system.

A HMM has been developed to find protein coding genes in *Escherichia coli* (*E.coli)* bacteria [Krogh, 1994] using the *deoxyribonucleic acid* (DNA) sequence from the EcoSeq6 database [Rudd, 1993]. This HMM includes states that model the codons (consisting of three amino acids denoted here as A, G, C, and T) and their frequencies in *E. coli* genes, as well as the patterns found in the intergenic region. The parameters of the HMM are estimated using approximately one million nucleotides of annotated DNA from the database and the model is tested on a disjoint set of about 325 000 nucleotides. The HMM finds the exact locations of about 80% of the known *E. coli* genes and

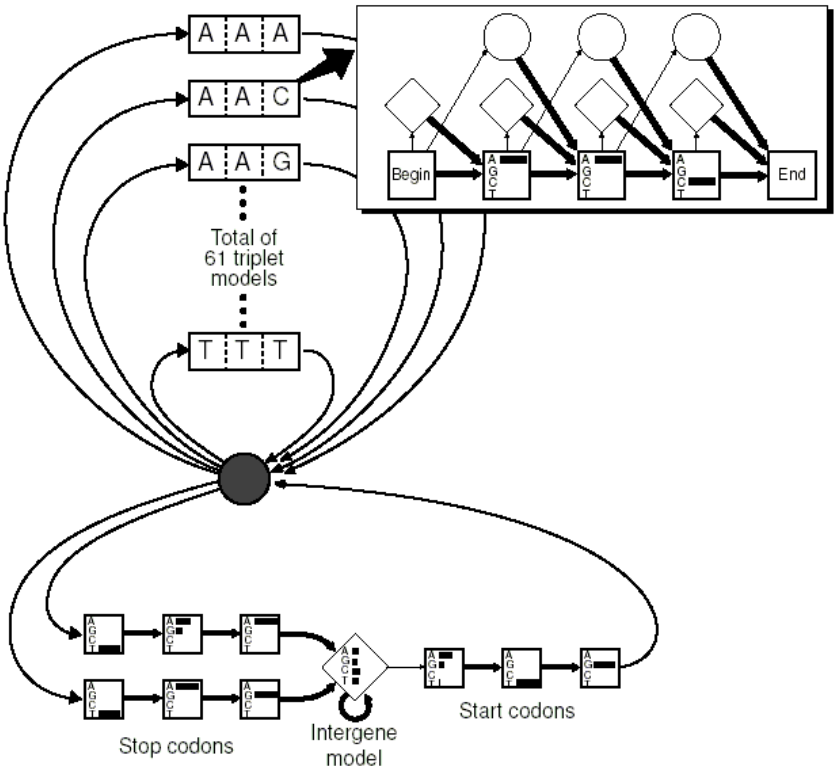

Figure 6.3. Hidden Markov model for recognition of *E. coli* genes (from [Krogh, 1994]).

approximate locations for about 10%. It also finds several potentially new genes and finds several places for possible insertion and deletion errors in the sequence.

A parser for *E. coli* DNA is presented in Figure 6.3. It consists of 61 sub-models for recognizing codons. The sub-model for an AAC codon in the Figure represents the common structure for each of these sub-models. The square nodes represent main states (one amino acid) and diamond nodes represent states where an additional amino acid can be inserted. Nodes denoted with circles can be used to represent deleted amino acids. The control state is used to connect all nodes and is represented as a black circle. Bellow the control state a sub-model for the intergenic region (part of a DNA between genes) is shown. It consists of nodes to recognize stop codons (a set of three codons which mark the end of a gene), a node to generate random sequence of codons between genes, and nodes for a set of three codons which always mark a start of a gene.

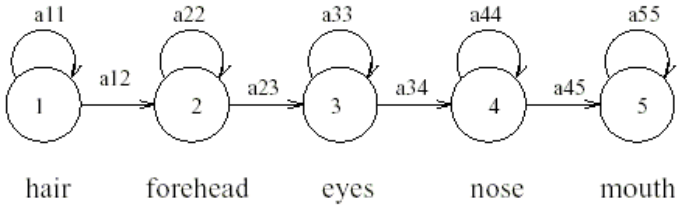

Figure 6.4. Hidden Markov model for face recognition (from [Nefian, 1998]).

Face recognition is an other active research area. In recent years it has become very popular, in part driven by increased security concerns. A number of techniques have been used, but their recognition rate decreases rapidly with changes in face orientation or image size. A face recognition system based on hidden Markov models has been developed [Nefian, 1998] and tested on a Olivetti Research Ltd. database of 400 images of 40 persons. Each person has been photographed with different face orientations, facial expressions and hairstyles.

In this system, the face image is divided into overlapping blocks, which represent hair, forehead, eyes, nose and mouth. A two-dimensional Discrete Cosine Transform (DCT) has been used to extract important features from each block. A window of size 13 by 3 over the lowest frequencies in the DCT domain has been chosen to represent the significant features for each block type. The observation vector therefore consists of 39 DCT coefficients. One left-to-right hidden Markov model with five states was trained to represent each individual in the image database as presented in Figure 6.4. A set of five images is used for each individual.

A set of 200 images, not used in training, were used to test the recognition rate of the system. Each image was divided into blocks and DCT coefficients were calculated. They were further applied to each of trained models and the model with the largest probability was selected. The system recorded recognition rates of 84%, which is significantly better than other recognition methods.

Speech recognition was among the first applications of hidden Markov models. SPHINX, a large-vocabulary, speaker-independent, continuous speech recognition system was developed at the Carnegie Mellon University [Lee, 1990]. The speech was digitised at 16 kHz and blocked into 20 ms frames. Various signal processing operations have been applied to frames, which resulted in 12 coefficients describing frequency

characteristics of the speech frame (known as *cepstral* coefficients) and the coefficient describing normalized power of the speech frame (relative to the sentence power). For both cepstral and power coefficient, the differences between coefficients for frame *n* and coefficients for frames *n*-2 and *n*+2 are calculated and used as additional coefficients.

The coefficients are converted into discrete symbols by using the Vector quantization method (described in more detail later in this Chapter). Early versions of the system used one codebook with 256 vectors, but results were not appropriate. In the later version of the system, authors used three independent codebooks, the size of each being 256. One was used for cepstral coefficients, an other for differenced cepstral coefficients, while the third was used for weighted combinations of power and differenced power coefficients. As a result, each frame observation consists of three symbols. Hidden Markov models used for recognition are modified to handle multiple observations.

The first version of the system used phonetic HMM's. Each phoneme (there are 45 phonemes in English) is modelled by a left-to-right hidden Markov model with topology as presented in Figure 6.5. Words (from the given vocabulary) are modelled by combining phoneme models according to the pronunciation dictionary. Sentences (from the predefined training and testing set) are modelled from the word models and silence models at the beginning and the end of a sentence (mandatory) and optional inter-word silence models. Such an approach did not provide satisfactory performance and the authors extended the architecture of the system by adding function-word-dependent phoneme models and generalized *triphone* models. Function words are particularly problematic in speech recognition as they are typically unstressed. Adding separate models for function word phonemes significantly improves the recognition rate. *Triphones* (context dependent phoneme model) are a popular approach to model

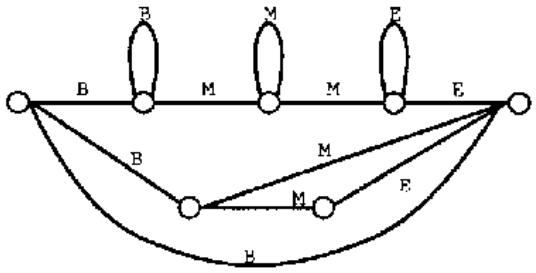

Figure 6.5. The topology of the phone HMM (from [Lee, 1990]).

coarticulatory effects in speach. However, there are many triphones which makes them hard to implement by HMM. In the SPHYNX system, triphones are clustered into 1000 generalized triphones.

The system was trained by using 4200 sentences from 105 speakers, and tested on 150 sentences from 15 speakers. A recognition rate of 96% was achieved with use of a simple word-pair grammar with transition probabilities estimated from the given set of sentences.

Navigation on the basis of previous experiences has been a research focus in the area of *autonomous mobile robots* for some time. Hidden Markov models were successfully used to recognize features of a corridor (angles, open doors, etc.) by using data from infrared and ultrasonic sensors mounted on a mobile robot [Aycard, 1997]. Recognized features are further used to determine the robot position in a given environment. The authors used second-order hidden Markov models. These models differ from the previously described ones by the fact that the state sequence is a second-order Markov chain. There, the probability of transition at time t depends on the states in which the process was in two previous time epochs ($t - 1$ and $t - 2$). The state transition probability matrix A is three-dimensional in this case, which requires changes in the training and evaluation formulas. Digitised signals from three ultrasonic sensors are used as observations. Signals were manually segmented and labelled. The authors selected three state left-to-right models for each of five places to be learned: start of corridor, end of corridor, open door, corridor and T – intersection. Places were generally well recognized, with the recognition rate of over 85%, except that the system had problems recognizing open doors. The authors further improved their method by using 16 sensors to recognize 10 different places, and by changing the data acquisition rate. The reported recognition rate increased to 92%.

Following these results, similar research has been done in the area of intelligent transportation systems. Nechyba and Xu [Nechyba, 1998] use hidden Markov models and data from a driving simulator to compare human control strategies. The system collects the values of the steering angle and longitudinal force on front tyres from the simulator, which runs at 50 Hz. Three state variables (angular, lateral and longitudinal velocities of the car) are also recorded. Data are blocked into overlapping frames and processed by fast Fourier transform (state variables) or fast Walsh transform (control

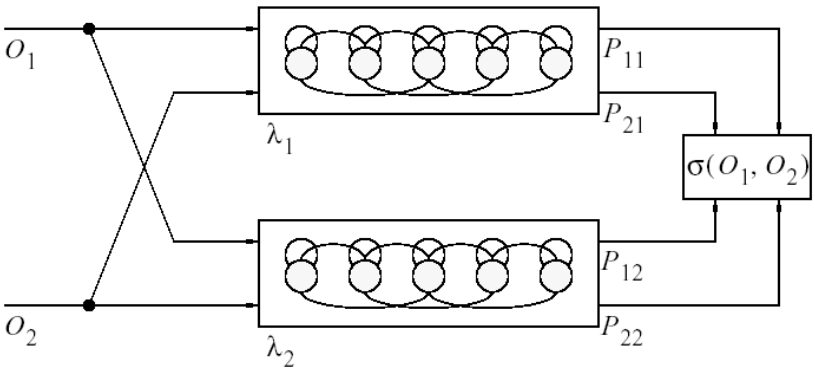

Figure 6.6. Four normalized probability values make up the similarity measure (from [Nechyba, 1998]).

variables). Calculated spectral coefficients are further converted to normalized power spectral density vectors. As the result each frame is represented as a set of 5 vectors (45 coefficients in total). Vector quantization is used to generate discrete symbols and a five state left-to-right hidden Markov model is used to model the particular driver.

For each driver, one hidden Markov model is trained, which we will denote by λ_j. Each observation sequence (O_i) is evaluated for all models and an appropriate probability is calculated $P(O_i, \lambda_j)$. The system, as shown in Figure 6.6., is able to successfully discriminate observations generated by various drivers by using a simple similarity measure given by the following formula:

$$\sigma(O_1, O_2) = \sqrt{\frac{P(O_2, \lambda_1)P(O_1, \lambda_2)}{P(O_1, \lambda_1)P(O_2, \lambda_2)}}$$

As we discussed in Chapter 3, the research team from MIT Media Lab has used hidden Markov models to model driver behaviour for a number of years. In [Liu, 1997] they described the system based on hidden Markov models for predicting driver intentions for a short time after the presenting the command for an action. Three state left-to-right hidden Markov models (as displayed in Figure 6.7.) are used to recognize driving intentions to execute the following actions: stop, left and right turns, line change, car passing, and car following.

The subjects drove a Nissan 240SX driving simulator. Data was recorded at 0.1 second intervals. For this experiment, the observation sequence consisted of the steering

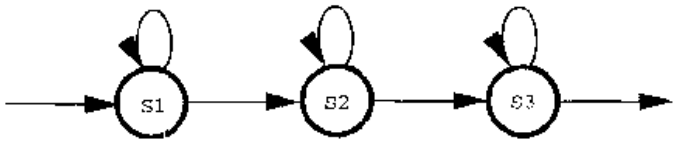

Figure 6.7. The three state hidden Markov models for driver intentions predictions (from [Liu, 1997]).

angle, steering speed and acceleration data. Text commands were presented to a driver a short time before the appropriate circumstances occurred (for example the vehicle arrives at an intersection). Recognition results were tabulated at one, two and three seconds after the presentation of a command to the subject. Mean recognition accuracies and standard deviations were 88.3 % ± 4.4 %, 89.4 % ± 4.3 %, and 87.5 % ± 2.8 %, respectively.

The same simulator has been used for experiments on continuous recognition of lane change events [Kuge, 2000]. In this experiment, drivers in the simulator are asked to change lane (to next lane right) or to stay in the current lane. In some cases they are forced to execute an emergency lane change by setting an obstacle in front (in the form of a large stationary track). The steering angle, steering angle velocity and steering force are measured. The architecture based on sub-HMMs has been developed for continuous recognition of events: lane keeping (LKN), emergency (LCE) and normal (LCN) lane change, as presented in Figure 6.8. The lane keeping event has been modelled as a single three state sub-model, while both lane change events are modelled as an ordered set of three sub-models. Each sub-model is further implemented as a single three state left-to-

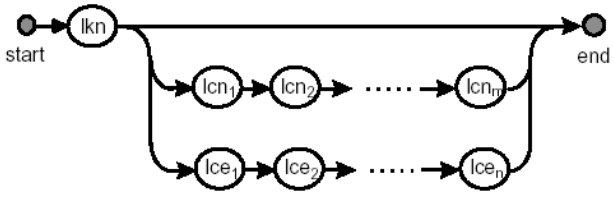

Figure 6.8. The architecture of the system for continuous recognition of lane change events (from [Kuge, 2000]).

right model. The authors do not provide any more details for selecting such architecture, in particular they do not justify why the combination of three sub-models is selected over a single model with more states.

The authors claimed the recognition rate of 98.3% for emergency lane change. Successful recognition is possible after 0.6 seconds and reaches its maximum at one second after the command presentation.

Further research in using hidden Markov models for recognizing driving manoeuvres at a tactical level is described in [Oliver, 2000]. The models are improved by incorporating elements of situation awareness. Information about car position relative to the road and lanes and the position of the driver's face and his eyes (gaze) is recorded by two external and two internal video cameras. No details about the hidden Markov models used for recognition are given. However, the authors show that the addition of road information and driver's gaze information generally increases the recognition rate of the models. The models are able to recognize the manoeuvre on average one second before any significant change in the car or any contextual signals take place. The authors also propose coupled hidden Markov models (CHMM) for modelling two interacting processes (the driver and the car). In CHMM, as presented in Figure 6.9., there are two independent sets of states (S and S') and two sets of observations (O and O'). The state of the each sub-model in the next moment depends on current states in both sub-models. The authors propose the use of asymmetric CHMM where the model for surrounding traffic affects the behaviour of the driver model, but not vice-versa.

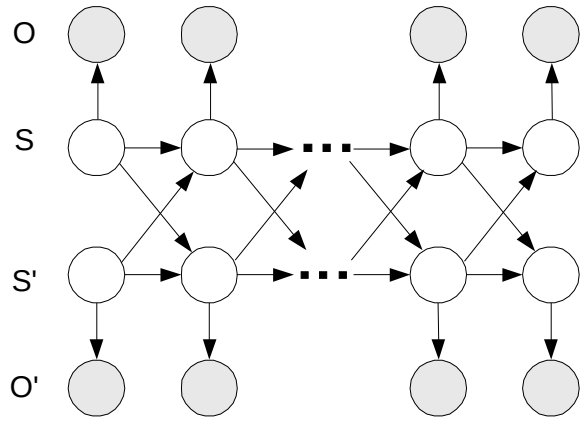

Figure 6.9. Coupled HMM rolled-out in time (from [Nuria, 2000]).

6.4. Data for Driving Event Recognition

As we discussed in previous chapters, the goal of our research was to provide a simple and inexpensive method for learning driving patterns. For experiments in driving pattern recognition, we decided to use longitudinal and lateral acceleration (because we use them in neural network experiments) and the vehicle speed. In order to maintain the feasibility of the real-time recognition system in the future, we decided to use discrete hidden Markov models, which are easier to implement and faster for execution than models with observations from a continuous domain.

Similarly to neural network experiments, the data from sensors must be processed before they are ready for use in training and evaluating the hidden Markov models. The use of discrete HMM requires conversion of continuous sensor data into discrete symbols from a predefined set. The complete process of transformation of continuous signals, from sensors into sequences of symbols, is presented in Figure 6.10. First, a

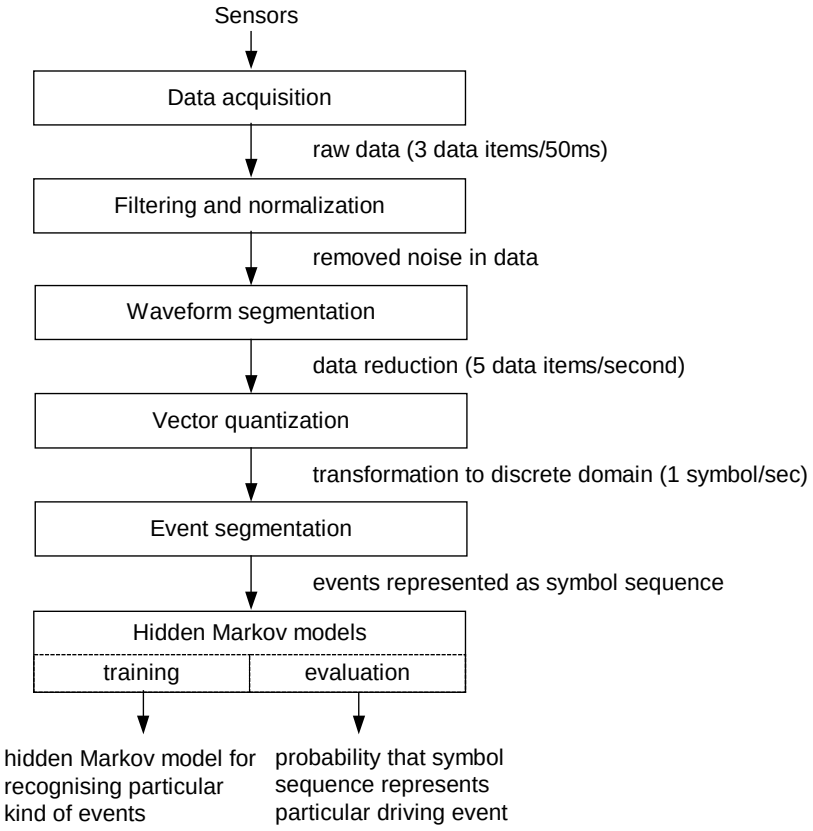

Figure 6.10. The process of sensor signal transformation into discrete symbol

number of processing steps are executed to prepare data in a continuous domain. Then, conversion from the continuous to the discrete domain is achieved by applying Vector quantization. In this Section we will describe each of these steps in more details.

6.4.1 Preparing data

The data acquisition system was described in Chapter 4. As we discussed there, the data received from sensors has a certain amount of unwanted noise that should be removed. We used a digital Butterwort low-pass filter with 2 Hz cut-off frequency for noise removal.

In these experiments we are not interested in the exact acceleration and speed values, but rather in patterns in data. For this reason, we preferred to work with normalized data. As a result of normalisation, the values for both lateral and longitudinal acceleration and speed are expressed in the range 0 to 1. Accelerations have positive and negative values, so we set a value of 0.5 to represent 0 *g* acceleration. Maximal encountered values for lateral and longitudinal accelerations (in all test drives) were increased by 10% and then used to calculate normalization parameters for both accelerations. The speed normalization range is selected to be 0 to 70 km/h, which is slightly above the highest recorded speed in all test drives (in test drives we used only streets with speed limits up to 60 km/h).

To reduce the amount of data we have to process, we decided to extract important features (shape descriptors) from the data collected from sensors. The waveform segmentation is a well-known technique for feature extraction from time series data [Pavlidis 1973], [Horowitz, 1975]. Here we use the least square approximation method described in [Bowerman, 1993]. Normalized data for lateral and longitudinal acceleration and speed are split into segments (frames) of predefined length. We selected a frame length of 11 data logging intervals. Frames overlap with each other, for one data-logging interval on both sides. Consequently, the effective frame length is equal to 10 data-logging intervals, as displayed in Figure 6.11. Therefore the effective duration of each frame is 10x50ms = 500ms. The frame overlapping is an often used technique to emphasize the influence of global trends in data.

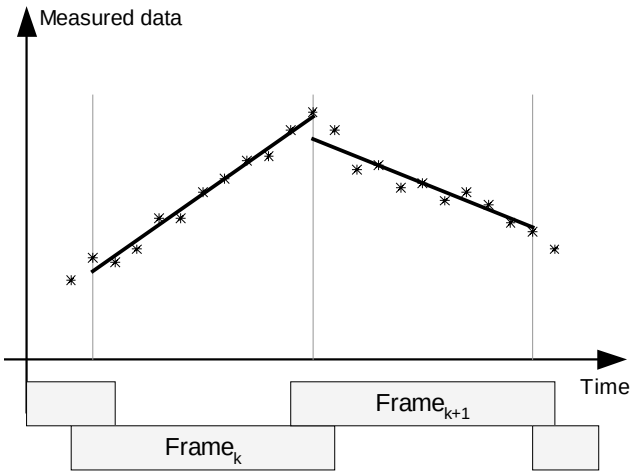

Figure 6.11. Overlapping frames and linear data approximation.

The least squares method for linear data approximation is used to calculate frame parameters, average (mean) value $\bar{y}$ and slope (change in values) k, by using the following formulas:

$$k = n * \sum_{i=1}^{n} x_i y_i - \frac{\left(\sum_{i=1}^{n} x_i * \sum_{i=1}^{n} y_i \right)}{n * \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2}$$

$$\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}$$

The actual implementation of this algorithm is simpler than these formulas, as data from sensors are collected in regular time intervals, which mean we can freely use the substitution $x_i = i$. Many sums in the first formula have a therefore constant value for each frame and do not have to be re-calculated.

As a result of waveform segmentation, each frame is represented by five parameters: average speed, average lateral acceleration, average longitudinal acceleration, lateral acceleration slope, and longitudinal acceleration slope. We do not need to use the speed slope value, as it has the same physical meaning as longitudinal acceleration. Figure 6.12. displays the segmented output of data collected from sensors

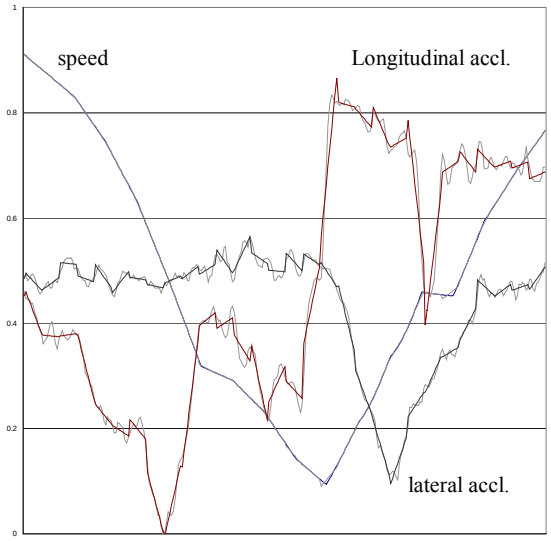

Figure 6.12. Results of the waveform segmentation for speed and lateral and longitudinal acceleration data.

during a right turn on an intersection. It is easy to note that all important waveform features are preserved, while the number of required data values has been reduced by a factor of six, since each frame is represented by 5 instead of 30 data values.

6.4.2 Vector Quantization

In order to use discrete HMMs we need to translate continuous domain frame parameters into symbols from a predefined set. *Vector quantization* is a standard procedure for joint quantization of a set of continuous-amplitude signals into one of the *codebook* (discrete-amplitude) values [Gray, 1984]. Vector quantization is widely used in data compression, telecommunication and speech recognition [Makhoul, 1985]. The vector quantization process in which we transformed the five frame parameters (longitudinal acceleration mean a_m and slope a_k, lateral acceleration mean b_m and slope b_k, and speed *v)* into a symbol *S* is presented in Figure 6.13. The vector quantifier module finds the entry in the codebook with minimal error and reads the appropriate symbol and sends it to the output.

The design of the codebook is a process of determining the set of vectors that will minimize the quantization error for a given data set. We use a well-known iterative

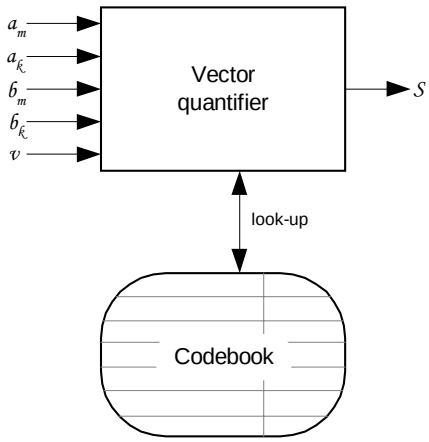

Figure 6.13. Vector quantization for five frame parameters.

clustering algorithm known as the *K-Means* algorithm. It starts with a random, uniform or otherwise initialised codebook. Vectors from the training set are then classified into a cluster using a predefined distance formula. New codebook vectors are calculated as centroids of the clusters. This iterative process continues until the total quantization errors decreases. As the K-Means algorithm is known to converge to the local minimum of the quantization error, it is repeated many times with different initial values for the code vectors in order to find the best possible global solution.

The selection of the codebook size is a tradeoff between smaller quantization error (for a larger codebook size) and easier HMM operations (for a smaller codebook size). Due to the stochastic nature of HMMs and the numerical problems described above, HMMs trained with larger sets of observation symbols had a worse recognition rate than HMMs trained with smaller sets.

In our case the input into the vector quantifier are vectors consisting of five parameters describing each frame. We used all collected data to create a large training set. The output is a stream of codebook indices (marked with letters `A, B, C,`…), which represent observation symbols for HMMs. We conducted a number of tests with various codebook sizes, ranging from 12 to 36. As could be expected, an increase in the codebook size reduced the quantization error, but the obtained HMMs had worse recognition rates. A codebook size of sixteen is selected as a convenient compromise. In Figure 6.14. we presented the normalized values for input signals and quantified values obtained as codebook vectors with minimal error. Even though we used a very small

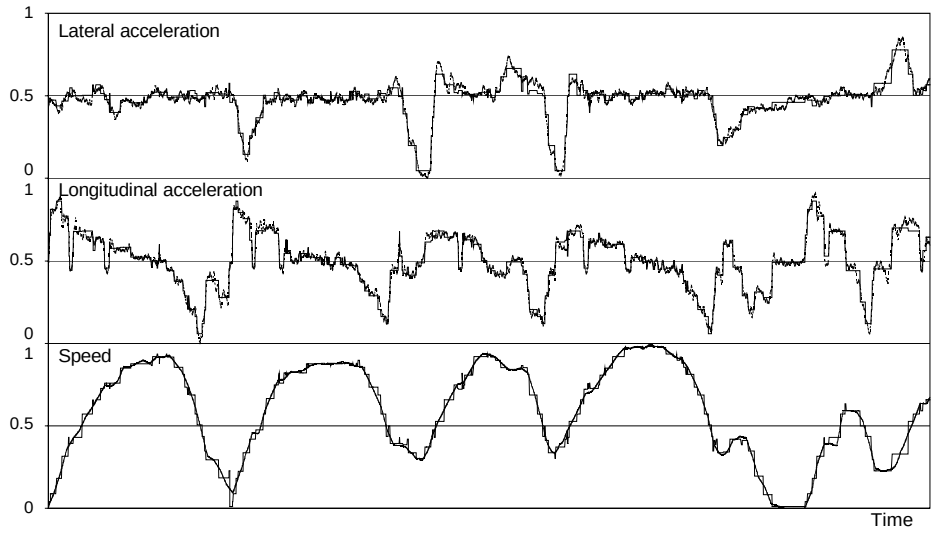

Figure 6.14. The original and quantified values for lateral (top) and longitudinal (middle) acceleration, and speed (bottom) for codebook size of 16.

codebook size (only 16), the total quantization error is small and all important signal features are preserved.

6.5. Driving Event Recognition by Hidden Markov Models

We use the term *driving event* to refer to any major change in vehicle attitude or speed, such as a left or right turn, a stop, making a U-turn and so on. From the decision making point of view driving events are demanding, as they require the driver to act on vehicle controls while, at the same time, the relationship between the vehicle and the environment (road infrastructure) is far more complex than between events. In these circumstances driving safety margins are very small, and even a minor disturbance can cause accidents. Accordingly, driving events have a very significant role in driving safety research.

Here we use hidden Markov models to recognize driving events from data obtained from a small number of vehicle sensors: lateral and longitudinal acceleration and speed. Recognized events could be used for different purposes in intelligent transportation systems. Later in this Chapter we will describe the use of recognized events for long term prediction of drivers' intentions.

For experiments with hidden Markov models we used data from 22 test drives around northwest Christchurch, undertaken during late 1999. No particular routes for test-drives were selected, but we tried to incorporate a variety of urban road features which could be found in medium-sized New Zealand towns. Some of the routes, or parts of routes, were repeated in order to provide data for our later experiments with driving event prediction.

Training and testing data for experiments with HMM were manually selected from data collected during test drives. A program has been developed for manual selection of data representing a driving event and marking it as a particular type of event. A screenshot of this program is shown in Figure 6.15. The original values for speed and lateral and longitudinal acceleration are graphically presented in the white area of the program window. The scroll bar could be used to browse to the required event. The user uses a mouse to select the period that represents a driving event. A black background highlights the selected period. The selection includes the whole number of frames. Once a user finished a selection by releasing the mouse button, a popup menu for event labelling becomes visible on the right of the selected data. The user labels the selected data by choosing one of the driving events from the popup menu.

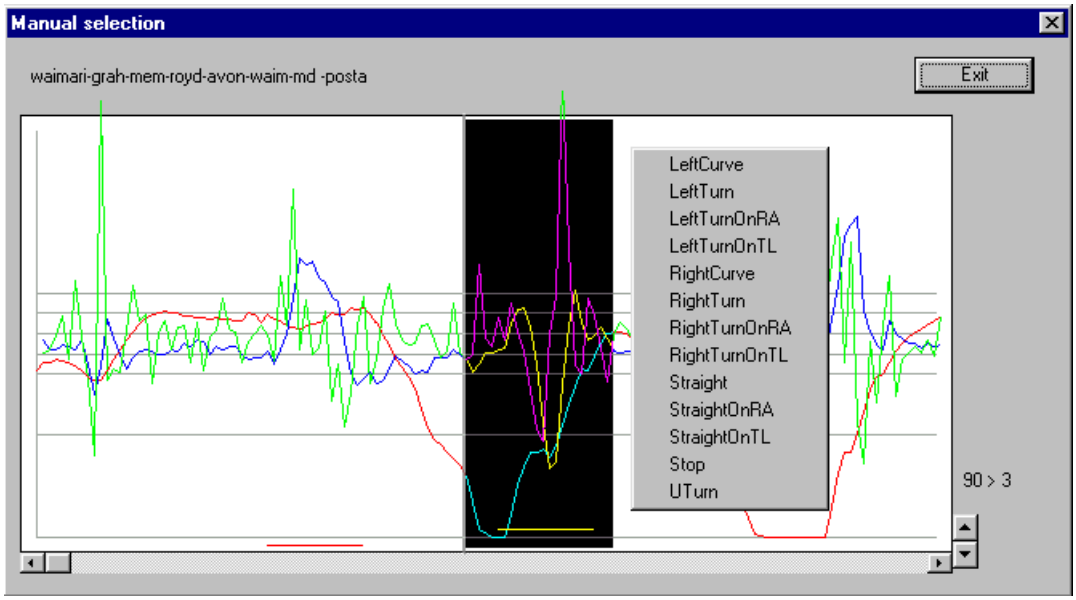

Figure 6.15. The program for selection and labelling of driving event.

We used this program to select and label 263 events from 22 test drives. As HMM training requires larger training sets, we used only seven types of the most common driving events in our experiments: driving a vehicle along left and right curves, turning a vehicle left and right on intersections, with and without roundabouts, and driving straight across an intersection with a roundabout. Roundabouts (rotary intersections) are very common traffic features in New Zealand urban areas. Each driving event is therefore described by its type, duration (expressed as number of frames) and the sequence of symbols (represents vector quantified data for each frame). The driving event data are stored in a Microsoft Access database, together with the symbolic and the numeric representation of all test drives. Storage into a database was chosen to facilitate search and manipulation of sequences. For example, one simple query operation was enough to retrieve all right turn driving events which are presented in Figure 6.16.

```
T 15: 9 8 8 8 8 8 8 8 2 2 3 1 1 12 4
T 14: 11 11 9 9 9 8 2 2 3 3 1 14 12 4
T 18: 11 9 9 16 6 9 8 8 8 8 2 2 3 1 1 12 12 13
T 20: 11 11 9 9 9 8 8 8 8 8 2 3 3 1 12 1 4 12 4 10
T 17: 4 11 11 11 11 6 3 3 1 1 13 13 12 12 4 10 4
T 15: 11 10 6 11 9 8 9 9 8 2 2 1 1 12 15
T 10: 11 6 6 3 3 1 1 12 4 10
T 11: 11 9 9 9 8 8 2 2 3 1 8
T 14: 11 11 11 9 9 8 2 2 3 1 1 12 12 13
T 13: 9 9 9 8 8 2 3 3 1 1 12 4 12
T 15: 4 4 11 11 9 6 6 6 3 1 12 12 12 13 4
T 16: 11 11 9 9 9 9 8 8 2 3 1 1 1 15 13 13
T 14: 10 11 11 6 3 3 3 1 1 12 13 10 12 4
T 22: 8 8 8 8 8 8 8 8 8 8 8 8 2 3 3 1 1 13 12 4 4 4
T 17: 8 8 8 8 8 8 8 8 8 2 2 3 1 1 12 4 12
T 11: 4 4 11 6 3 3 1 1 12 12 12
T 11: 4 11 11 11 6 3 3 1 1 12 12
T 15: 11 11 11 6 3 3 3 1 12 12 13 10 4 4 10
T 13: 11 9 9 8 2 3 1 12 12 12 12 4 10
T 11: 12 4 11 11 11 6 3 1 1 1 12
T 13: 9 9 8 2 2 3 3 1 1 12 4 10 4
T 14: 11 11 11 6 6 3 1 1 1 12 4 12 4 4
T 12: 9 6 9 9 8 2 3 1 1 12 12 13
T 14: 11 9 9 9 9 8 2 3 3 1 12 10 12 13
T 11: 12 13 6 6 3 1 12 1 12 12 12
T 9: 11 11 6 3 3 1 1 12 12
T 14: 11 11 9 9 9 8 2 3 1 1 12 13 10 12
T 16: 11 11 11 11 6 6 3 1 1 10 12 4 12 10 10 4
T 9: 9 9 9 9 2 2 3 1 1
T 12: 8 8 2 2 3 1 1 12 12 10 10 4
T 7: 9 8 3 3 1 8 12
T 14: 11 11 11 11 11 9 6 3 3 1 1 14 13 10
T 9: 8 2 2 1 1 1 12 14 12
T 10: 8 8 2 2 1 1 12 14 12 12
T 6: 8 2 2 1 1 12
T 10: 8 8 15 5 2 3 1 1 12 10
```

Figure 6.16. The list of all right turn driving events. Each event is represented by a number of symbols, followed by codebook indexes representing symbols.

The above program additionally collects statistical data regarding all driving events, such as minimal and maximal duration of the events, minimal and maximal values for signal data for each type of event, etc. We used these data for driver event prediction as described later in this Chapter.

We constructed seven HMM models - one for each type of driving event we wish to recognize. As mentioned above, we used left-to-right discrete hidden Markov models as they are know to work well with temporal data sequences. We made small preliminary tests with various model sizes (from five to eight states) in order to find an optimal model for each driving event. If the number of states in the model is too small, the false rejection error rate increases, as the model is not flexible enough to cover all events of a particular type. On the other hand, as presented in Figure 6.17., a model with too many states leads to a larger rate of false acceptance errors.

To find optimal model sizes for each event type we constructed models with sizes 4 to 8. Models were trained by representative subsets of sequences for the given event type. Then observation probabilities were found for each observation sequence in the training set. We selected the model size that had the smallest variance in output probabilities. In all cases, except for left turns, we selected HMMs with six states. For left turns, we selected model size five, since average lengths of observation sequences for left turns is smaller than for other types of driving events.

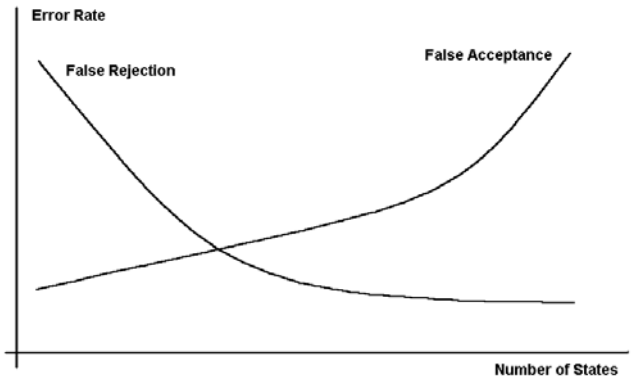

Figure 6.17. The effect of the number of states to the recognition error.

Each model was trained with observation sequences from the training set for the particular type of event. For training we selected representative subsets of events, which consist of approximately 30% of all events of a particular type. The training sets for events with smaller number of sequences had a proportionally higher percentage of events to provide a reasonable amount of training data. This selection was also done manually, based on the visual inspection of symbol sequences.

For training we used the Baum Welch algorithm. We tested a Viterbi-based training algorithm as well. However, the results were inappropriate, probably due to the limited number of the training sequences. Data for our HMM experiments consist of a number of short observation (symbol) sequences (5 to 30 observations per sequence), which are grouped according to the type of driving event they represent. A small number of observations in the training set, and an observation probability distribution that is far from uniform, leads to typical numerical problems encountered in HMM implementations. The forward and backward variables are computed by multiplying probabilities, and in such a circumstance they could easily become smaller than the computer's numeric precision. In order to prevent this forward variables are re-scaled after each iteration. The inverse scale factor is applied to backward variables in order to cancel the effect of scaling on re-estimation formulas.

Small observation probabilities could cause the observation probability for some observation symbols to become zero. This effectively removes some transitions by setting the transition probability to zero. Due to the re-estimation nature of training procedures, every removal is permanent and cannot be repaired in the next iteration. In order to avoid this problem a simple flooring method is implemented. If, after re-estimation, we have $\bar{b}_{jk} = 0$, we replace it with a small value ε. This simple approach prevents numerical problems without compromising the calculated probability. More complex algorithms exist [Dai, 1995], but we found flooring quite appropriate for our application.

Due to short observation sequences each training iteration must include all available training sequences, or otherwise many re-estimated parameters become zero. Therefore we use modified re-estimation formulas for training by multiple observation sequences [Yang, 1995]:

$$\overline{a}_{ij} = \frac{\sum_{k=1}^{Q}\sum_{t=1}^{T-1}\alpha_t^k(i)a_{ij}b_j(O_{t+1}^k)\beta_{t+1}^k(j)}{\sum_{k=1}^{Q}\sum_{t=1}^{T-1}\alpha_t^k(i)\beta_t^k(i)}, \qquad \overline{b}_{jk} = \frac{\sum_{k=1}^{Q}\sum_{\substack{t=1 \\ O_t=V_k}}^{T-1}\alpha_t^k(j)\beta_t^k(j)}{\sum_{k=1}^{Q}\sum_{t=1}^{T-1}\alpha_t^k(j)\beta_t^k(j)}$$

After each run of the Baum Welch training algorithm we noted the calculated model parameters. In order to find a better global solution we ran 30 iterations of the Baum-Welch algorithm with different initial parameters. The model with the highest probability was chosen as the model for the particular driving event.

For model evaluation we used the procedure for forward variable calculation described above as part of the Baum Welch algorithm. In the beginning we evaluated each trained model in isolation. Firstly, we calculated probabilities for all observation sequences from the data set for the event type for which the model was trained. We used the resulting probabilities to determine the threshold value. Then we calculated probabilities for each observation sequence in all test data sets. If the probability was above the threshold, this particular driving event was marked as an event of the same type as the model. The results of these tests are summarized in Table 6.1. Each row in the table represents the number of sequences positively recognized as events of a given type. In an ideal case all cells in the table should be zero, except for the diagonal values, which should be equal to the number of test data.

	Evaluated sequences (the total number of sequences for the particular type of event)						
Recognized as	RC 41	RRA 16	RT 36	LC 56	LRA 20	LT 59	SRA 10
Right curve (RC)	41	0	0	6	0	0	0
Right on roundabout (RRA)	0	16	6	0	0	0	0
Right turn (RT)	1	7	36	0	1	0	3
Left curve (LC)	10	0	0	55	0	1	0
Left on roundabout (LRA)	0	0	0	0	20	26	0
Left turn (LT)	0	0	0	1	17	57	0
Straight on roundabout (SRA)	0	1	12	0	3	0	10

Table 6.1. Driving event recognition by isolated HMMs.

Recognition by an isolated HMM is the simplest approach, but results obtained by experiments show that, in most cases, driving event recognition was accurate. Isolated HMMs almost always correctly recognized events of the type for which they were trained. Also, for most pairs of distinct events, the result is zero, which confirms that the HMM very reliably differentiates between events of these types. However, there are four pairs of events for which isolated models have problems discriminating between:

- Right turn vs. Right on roundabout,
- Right turn vs. Straight on roundabout,
- Right curve vs. Left curve, and
- Left turn vs. Left on roundabout.

In order to improve recognition performance we implemented a parallel evaluation of observation sequences by multiple HMMs. From the complete set of 238 sequences, each observation sequence is fed into all seven trained models, as presented in Figure 6.18. The computed probabilities for all seven models are sent to an arbiter. The arbiter selects the highest of these probabilities (P_{max}) and determines the driving event type to which a sequence belongs. The results of the parallel sequence evaluation are presented in Table 6.2. Each row in the table represents the number of sequences correctly recognized as events of a given type. Using this approach, the models successfully

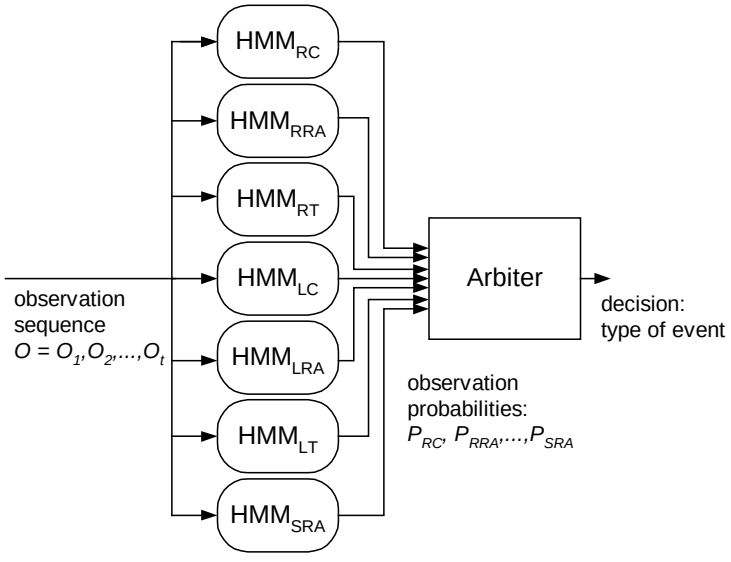

Figure 6.18. The parallel recognition of driving events.

recognized all events except the four cases (of 56, or 7.1%) of left curve events recognized as right curve events. This problem was caused by a relatively small codebook size used (16), which creates a very coarse vector quantization grid. Due to the coarse grid, small left and right lateral accelerations are clustered together. Consequently, the vector quantization algorithm is not able to distinguish between the cases when only difference in data is the sign of the (small) lateral acceleration. If required, this could be easily rectified by adding domain specific knowledge to the k–means algorithm used for the codebook learning, which would prevent such clustering .

	Evaluated sequences (the total number of sequences for the particular type of event)						
Recognized as	RC 41	RRA 16	RT 36	LC 56	LRA 20	LT 59	SRA 10
Right curve (RC)	41	0	0	4	0	0	0
Right on roundabout	0	16	0	0	0	0	0
Right turn (RT)	0	0	36	0	0	0	0
Left curve (LC)	0	0	0	52	0	0	0
Left on roundabout	0	0	0	0	11	18	0
Left turn (LT)	0	0	0	0	9	41	0
Straight on roundabout	0	0	0	0	0	0	10

Table 6.2. The results of the parallel recognition of driving events.

Furthermore, in 29 cases (of 79, or 36.7%) the models could not distinguish between left turns and left turns on a roundabout. However, the geometry of intersections, and traffic rules for left turns on intersections, with or without roundabouts are the same, so this result is not surprising and we believe that these two event types should be regarded as one.

If we exclude the above problem, we can conclude that, even with a very limited set of sensors, the system correctly recognized 234 of 238 (or 98.3%) driving events. The difference between the highest (P_{max}) and the next highest (P_{max2}) probability calculated for an observation sequence was large. For example, the average difference

calculated as: $\frac{\sum_{i=1}^{N} \frac{P_{MAX} - P_{MAX2}}{P_{MAX}}}{N}$, for right turns with and without roundabouts (which are reasonably similar events), was above 48%. That means that the probability for the correct model was almost twice the probability for the next best model. This proves that the presented model for driving event recognition is not only very accurate, but also reliable and robust.

We believe that the recognition rate achieved is sufficient for the applications described in the previous chapters. We expect the recognition rate to further increase as better training sequences become available, providing a greater variety of training data for each driving event. If required, the recognition rate could be easily improved by adding new sensors (using GPS positions for example), or by employing the domain specific knowledge in the vector quantization.

The recognition process consists of a few steps. However it is designed to reduce data and maintain the simplicity of required numerical operations. Consequently the processing power required for real-time recognition is far lower than what is available in modern CPUs. The training of hidden Markov models in a real-time environment could be more demanding due to its iterative nature. However, training does not have to be executed after each new event and the amount of training data could be kept at a reasonable level by only selecting representative event subsets, as described for system ALVINN, in Chapter 5.

6.6. WinHMM, a Program for Experimenting with Hidden Markov Models

Experiments with hidden Markov models require a stable and flexible software tool. This tool should enable the easy manipulation of training and test data, and the appropriate visual representation of model parameters and evaluation results. When we conducted our experiments in 1999, there were not many software packages available for experimenting with hidden Markov models. Commercial software was out of reach given our research budget. Even the HMM software developed by academic institutions, like the HTK toolkit developed at Cambridge University [HTK, 2002] and used by

Pentland's research team [Pentland, 1995], were only commercially available. On the other hand, available public domain software was only implemented in fragments and ported to platforms used by the public-domain programming community (i.e. Unix and the GNU C compiler).

For this reason we decided to develop WinHMM, a program for training and evaluation of hidden Markov models. In its design we started from the public domain source code developed by Dr. Tapas Kanungo, at the Institute for Advanced Computer Studies, of the University of Maryland, [Kanungo, 2000]. This code was developed in standard C, and implements the basic hidden Markov model algorithms such as: Forward-Backward, Viterbi, and Baum-Welch. It also implements basic data structures (vectors and arrays), and basic input-output functionality for model parameters and training and testing data.

By identifying important concepts and modelling classes we redesigned the existing code into a simple object-oriented model. These classes include low-level numerical entities like vectors and matrices, concepts from hidden Markov model theory (like model and observation sequences) and high-level entities such as documents (which are containers for application data: models and observations) and views (which produce the visible output from the program). The original code is re-implemented to support such a model and ported to Microsoft's Visual C++ programming language. We also improved existing algorithms to handle numerical problems due to a limited number of training sequences; as described in the previous section.

We also provided a simple but flexible user interface for easy manipulation and visualization of the data, as presented in Figure 6.19. The user creates a new model by specifying the topology (currently only selecting between left-to-right and general topologies) and entering the number of states (the presented model has six states) and the number of observation symbols (sixteen in this example). In the upper part of the WinHMM's application window, the (incomplete) diagram of the selected topology is shown. It is also possible to open an existing model from a file, which is created by the WinHMM's Save function.

When a model is created or loaded, the user may choose a training algorithm. Currently, WinHMM supports the Baum Welch and the Viterby training algorithms. Training starts when the user selects a file with observation sequences. The structure of

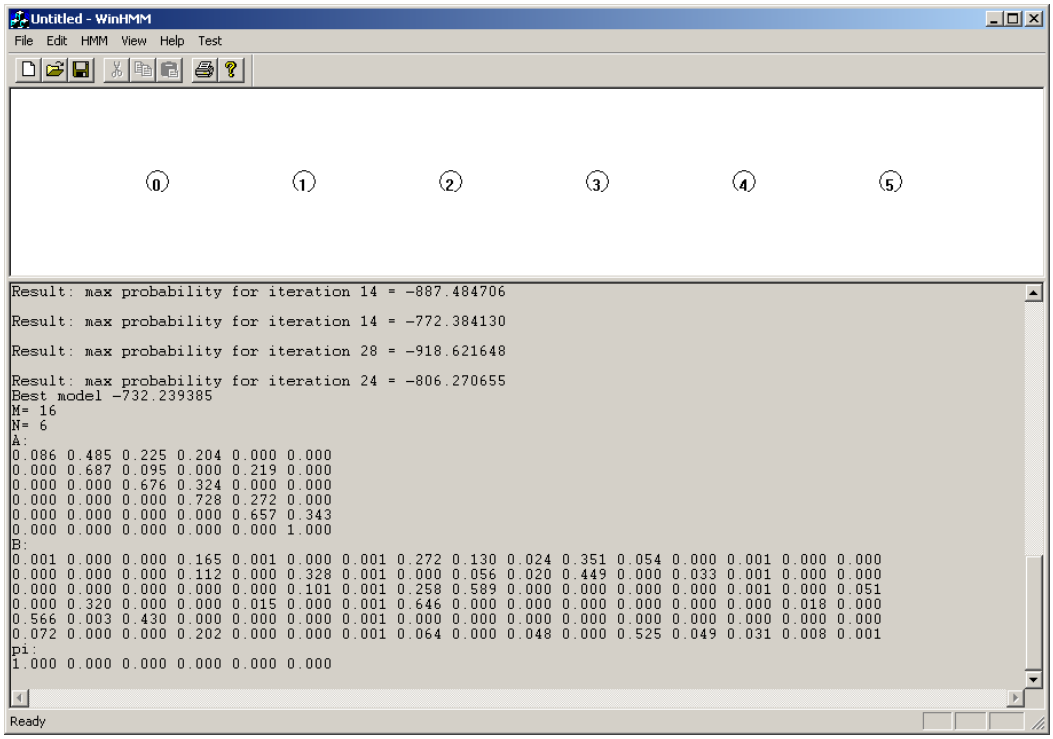

Figure 6.19. The model resulting from training in the WinHMM program.

an observation file is summarized in Figure 6.16. The implementation of the training algorithms follows the procedure described in Subsection 6.2.3, and Section 6.4. The described training algorithms are repeated the fixed number of times, and the model with the minimal probability is selected at the end of the training. The lower part of the WinHMM's application window in Figure 6.19. shows the results of the last few repetitions of the training algorithm. After the result of the last iteration, the probability for the chosen best model is given, as well as parameters for the model (denoted with: M, N, A, B, pi, in the WinHMM's output window). The WinHMM system uses the logarithms of the probabilities in calculations, as described in the previous Section, and such values are presented here instead of direct probability values. Once the best model is obtained the user may save it into a file for later evaluation.

The evaluation process is very similar to the training process. The user opens the previously trained model, which is saved into a file, and chooses the evaluation algorithm (Forward variable calculation or the Viterby algorithm). Then he/she selects the test file with observation sequences and the WinHMM calculates probabilities for

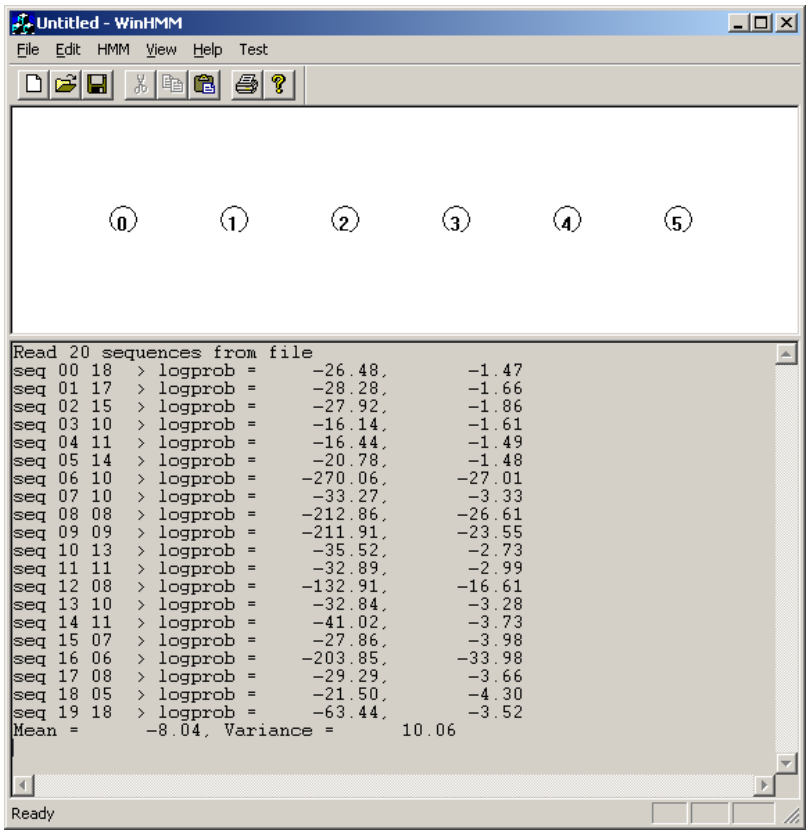

Figure 6.20. Evaluation of the test data set.

each sequence. The output from the WinHMM evaluation process for recognizing right turns in the test file consisting of twenty sequences is presented in Figure 6.20. The first six sequences in this set are right turns from the training set, while other sequences represent other event types.

The Arbiter module for parallel recognition is implemented by a simple Excel macro. Through the Windows Clipboard tool we are able to copy the output from the WinHMM output window to an Excel worksheet. By evaluating all test sequences on all seven models (for seven different driving events we are interested in), we obtained all necessary data to select the most appropriate type of driving event for each sequence. The Arbiter macro simply compares the resulting probabilities, selects the largest, and pronounces the type of the driving event for each sequence.

The WinHMM Software developed for our experiments has been also successfully used for event recognition in video streams at the University of Twente, as described in [Petkovic, 2003].

6.7. Long Term Prediction of Driving Events

As asserted in Chapter 3, identifying and learning patterns in driving can generate useful information, which can help the driver perform a range of different navigational driving tasks. In order to validate the learned driving patterns we use them to predict future driving events. In previous sections we showed that we can easily and reliably recognize driving events, which is the first step towards identifying and learning driving patterns. In this Section we describe a method for driving event prediction and the results of event prediction based on real driving experiences.

The system, described in the previous Section, recognizes driving events, which are related to major changes in a vehicle's attitude, such as left and right turns on intersections with and without roundabouts, driving along left and right curves, and driving straight on intersections with roundabouts. There are also other important driving episodes, such as driving along straight road segments, line-changes, U-turns, parking operations, as well as pauses in driving. We believe that some of these events, such as line–changes and U-turns, could easily be recognized, using the above recognition technique; and this is partially validated in related papers [Kuge, 2000]. Other driving episodes**,** such as driving straight and periods while a vehicle is stationary, are also not included in our recognition experiments since we believe that they could easily be distinguished by processing data from sensors. Driving along straight street segment, for example, can be detected from the driving data (between recognized driving events) by taking segments of driving data where lateral acceleration is below a predefined threshold. These are important elements of the driving experience and we will regard them the same way as any other types of events.

Periods where driving speed is close to zero (and GPS receivers are good in detecting stationary state) will be separately marked. We will distinguish two situations in which the vehicle could be stationary. When the duration of data with speed close to zero is small (say, less than five minutes), we will assume that the vehicle is stationary due to traffic regulations (pedestrian crossing, traffic lights, railway crossing, etc), and that this does not represent the start or end of a driving session.

When the duration of the vehicle's stationary state is above a predefined threshold, we will assume that the driver went outside his/her car and finished this driving session.

Such a location is significant for the driver and the number of such places, for an ordinary driver, is limited. For the purpose of clarity in further discussion we will introduce two virtual events: i.e. Start and Stop driving events which denote the moments when a vehicle starts moving from a place where the vehicle has been stationary for a longer period of time, or when a vehicle finishes its motion at such a place. In this experiment we are not interested in the exact physical location of these places, but the system may easily be used to infer the position relative to the recognized driving events. Also, a logical map of the street network used by the driver could easily be constructed.

The type of a driving event is important, but not sufficient to enable prediction based on previous experience. Events must be more precisely defined by specifying their duration and a number of other parameters (depending on the type of the event). The event definition process is presented in Figure 6.21. The continuous stream of frame parameters is generated from sensor data as described in Section 6.4. Frame parameters contain the description of the segmented waveforms in the sensor data. As discussed earlier, each frame is defined with the average speed, and average values and slopes for lateral and longitudinal accelerations. The stream of frame parameters is given as input to the *Vector quantization* module and the generated quantified symbols are further passed to the *Event recognizing* engine based on hidden Markov models. The same stream of frame parameters is given as input to the *Event definition* module. The Event definition module serves two purposes: it recognizes event types for straight driving and stationary states, and calculates required parameters for event definition.

The Event recogniser module supplies information about the type, starting time and duration of recognized driving events. Based on that information, the Event definition module detects periods with small lateral acceleration between already known

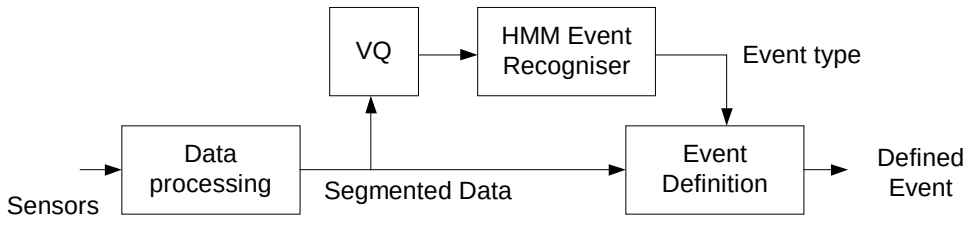

Figure 6.21. The event definition process.

events. These periods will be marked as straight driving events. Periods with speed values equal to zero will be also detected, and appropriate events will be generated.

The Event definition module uses values for frame parameters to calculate other important parameters for events recognized by itself and events recognized by hidden Markov models. Which parameters are calculated depends on the type of event. The list of event types and their respective parameters are given in Table 6.3.

Parameters recorded for events are very simple, such as duration of the event (in seconds), or total path travelled by the vehicle during the event (in meters). They are easily calculated from the frame parameters. For example, the length of the path

Event type	**Parameters**				
Short break	d				
Driving straignt	d	l	v_{max}	σ_v	
Left turn, incl. Left turn on RA	d	$\int_t a(t)$	l		
Right turn	d	$\int_t a(t)$	l		
Right turn on RA	d	$\int_t a(t)$	l		
Straight on RA	d	l	$\int_t \lvert a(t) \rvert$		
Left Curve	d	l	$\int_t a(t)$	v_{max}	σ_v
Right Curve	d	l	$\int_t a(t)$	v_{max}	σ_v

Table 6.3. Types of driving events and parameters used for event definition. Parameters are: d – the duration of the event, l – the length of the path travelled during the duration of the event, v_{max} – the maximum speed recorded during the event, σ_v - the standard deviation for speed during the event, $\int_t a(t)$ -the sum of lateral acceleration values during the event.

travelled is calculated by summing a speed value for all frames which are included into the event sequence and then multiplying by the duration of the frame. It could be argued that some other parameters are equally important for describing some event types. However, the results of our experiments showed that the parameters we selected provide good definition for events. In the table 6.3., the parameters are ordered by the increasing importance ("weight") which will be discussed later.

As a result of the event definition process each test drive is represented as a sequence of event objects. Each event object contains the type of the event and relevant calculated parameters. The data structure in which we store previous driving experiences used in driving event predictions is presented in Figure 6.22. This data structure is a dynamic list of driving sequences. Each sequence is also represented by a dynamic list of events. The virtual start and stop events are depicted in the figure as squares, while all other type of events are drawn as circles. All driving events representing driving along straight street segment are indexed by using a *Straight segment index* vector. Each entry in the index vector contains the value of the length parameter for the respective event, and a pointer to it. To enable a fast search the index vector is sorted by increasing length.

Currently we have only a relatively small number of driving experiences to store in the above data structure, and its performance seems adequate. However, with an increase in the number of driving sequences the proposed structure is likely to become

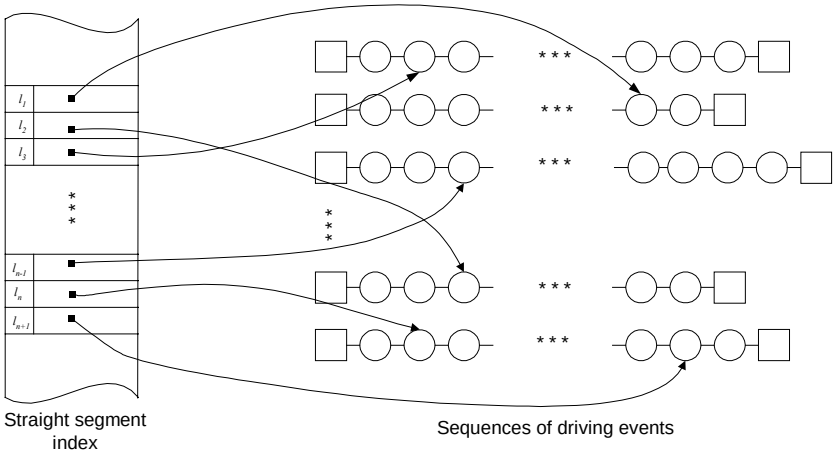

Figure 6.22. The data structure used for driving event prediction.

inefficient. Clustering similar parts of the sequences and converting a two-dimensional list into a network type structure could easily overcome this problem. Similarly clustering should be implemented for Straight segment indexes, which may have multiple pointers for some length values.

The architecture we designed and implemented for driving event prediction is depicted in Figure 6. 23. Inputs into the system are driving events, recognized and defined as described above. The *Predictor* module is the main part of the system. It evaluates each new event, and searches for the most likely next event. In order to predict a future event, the Predictor generates a number of *hypotheses*, sequences of events that are similar to the currently detected sequence. The *similarity measure* is the value of correlation between the sequence of events in a hypothesis and the currently detected one. The *hypothesis list* maintains all valid hypotheses, sorted by their similarity measures. For each hypothesis, we store the sequence of matched events, the similarity measure, and a pointer to the next event in the sequence from which the hypothesis is generated.

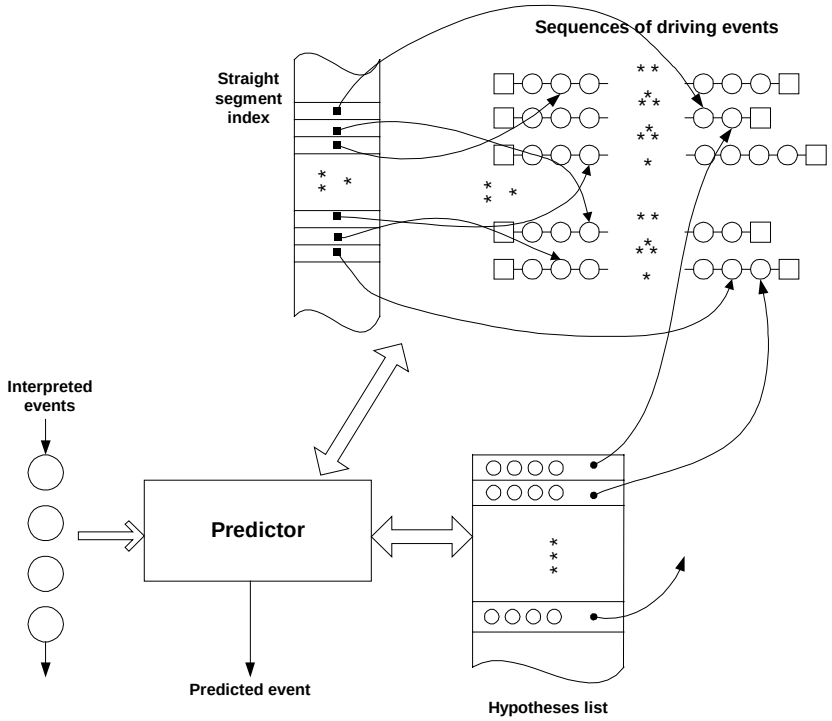

Figure 6.23. The architecture of the system for driving event prediction.

For each new input element the Predictor executes the following procedure:

- If the driving event is first after the start of driving, the predictor generates hypotheses for all driving sequences that start with the event of the same type and similar properties.
- If the event is defined as driving along a straight street segment, the Predictor also generates hypotheses for all similar events from the index, which are not covered by the existing hypotheses.
- For all existing and newly generated hypotheses, the similarity between the new input event and the next expected event are calculated and new similarity measures for all hypotheses are derived. The methods for these calculations will be described later.
- Hypotheses that have similarity measures bellow predefined thresholds are discarded from the list.
- Hypotheses which have very similar sequences of matched events, will be clustered together and the resulting hypothesis will have its similarity measure set to the highest measure increased by a small value proportional to the total similarity measures for all clustered hypotheses. In this way the system gives preference to sequences of events which are encountered more often.
- Pointers to expected events are updated for all remaining hypotheses.

As a result of the above process, we get a list of one or more hypotheses. The predicted events are referenced by hypotheses. The event which is referenced by the hypothesis with the highest probability is the most likely event to occur, and the Predictor module will declare it as the predicted event.

Short time break events, like stops on traffic lights or train crossings, could cause two physically identical routes to be regarded as two different ones. Some may argue that these events should not be taken into account, and that these sequences should be treated as two instances of the same event sequence. However, we believe that driver's behaviour could be influenced by such short stops and that this may be further useful to detect differences in subsequent events. Keeping these sequence as separate experiences will provide additional flexibility for the system, particularly when the number of previous experiences becomes sufficient.

Similarity between events of the same kind is calculated by comparing their parameters, using the following formula:

$$S(a,b) = \sum_i w_i s(p_{a,i}, p_{b,i}),$$

where *S(a,b)* denotes a similarity between events *a* and *b*. Here w_i is the relative importance ("weight") of the parameter *i* for this type of event, $s(p_k, p_l)$ is the similarity between parameters *k* and *l*. With $p_{a,i}$ we denote a value for the i^{th} parameter of the event *a*. The similarity between parameter values p_k and p_l is calculated using the formula:

$$s(p_k, p_l) = p_{min}/p_{max}$$

where p_{min} is the smaller of parameters p_k and p_l, while p_{max} is the larger of these two. If parameters p_k and p_l have similar values, their similarity will be close to one. The sum of all weights is also set to one, so in the case where all parameters for two events are equal, the similarity measure will be equal to one. For example, for measuring similarity between two events of driving along a straight road segment, the weights are 0.4 for duration and length, 0.15 for maximum speed, and 0.05 for speed variation.

Similarity between events of different types is generally zero, but is allowed in some cases, such as between right turns on intersections with and without roundabouts. For each of these cases we use additional formulas for similarity calculation.

Similarity between sets of events (such as between two hypotheses or between the recent event history and a hypothesis) could be calculated by using a similar model, expressed by the formula:

$$Z(A,B) = \sum_t u_t S(a_t, b_t),$$

where *Z(A,B)* is the similarity between sequences *A* and *B*, and *u* denotes weights. These weights are predefined to give preference to the most recent events. For example, if we compare event sequences of length 4, the weights will be $u_t = 0.4$, $u_{t-1} = 0.3$, $u_{t-2} = 0.2$, $u_{t-3} = 0.1$.

We use a simple iterative method to calculate the similarity of each hypothesis to the currently detected sequence of events, denoted as Z_{Ht}. When a hypothesis is created, the similarity between the detected event and a similar event from the selected driving

event sequence is taken as the initial value. A reinforcement-based approach is used to update the hypothesis' similarity measure for each new event. The previous value is updated by the similarity measure between the currently detected event and the event pointed to by this hypothesis, as in the following formula:

$$Z_{Ht} = \begin{cases} S(a_0, b_0) & t = 0 \\ i * Z_{Ht-1} + (1-i) * S(a_t, b_t) & t > 0 \end{cases}$$

The parameter *i* is an *inflation rate*. After some trials we set this at 0.65. In the current implementation it has the constant value, but its value may change if a finer control is needed. The hypothesis' similarity measure closely follows the calculated similarity measure. After just two pairs of very different events it may drop below the threshold level. However, it may "survive" a single mistake in the event recognition engine and quickly restore itself to the appropriate level. In situations where the similarity of hypotheses is high for a number of consecutive events, we may add further re-enforcement to the prediction probability, to reflect our increased confidence. In this experiment we add a *confidence factor* to the hypothesis' similarity measure, using the following formula $Z_{Ht} = Z_{Ht} + (1\text{-}Z_{Ht})(T/50)$, where T is the number of previous events for which $Z_H > 0.80$ and $S(a,b) > 0.65$.

We used the same set of 22 test drives that we used for driving event recognition experiments to test the above procedure. The list of test drives is given in Table 6.4. Start and end locations of each test drive are denoted using a two-uppercase-letter abbreviation (PS = Post Shop, OF = Office, etc.), while streets travelled are denoted by three letter abbreviations (mai = Maidstone Road, avo = Avonhead Road, etc.).

The map of all drives is presented in Figure 6.24. This map was created by using the ArcGIS software. We developed a macro for this software through which we loaded positions collected during these drives. These positions are stored into an ArcGIS geodatabase. Each test drive was organized as one polyline feature in the "Drives" feature class in the geodatabase. We used the Enveronmental Canterbury web map server (www.ecan.govt.nz) to generate the background image. This image is created from aerophoto images and vector data for streets and streams.

P	FR-avo-wty-cor-bro-pks-sol-mai-PS
P	OF-enr-mai-avo-mer-wit-mai-NH
P	NH-mai-wit-avo-roy-wpd-she-wai-aor-brt-ilm-bry-jef-cly-ilm-aor-bat-ilm-brt-OH
P	NH-mai-avo-gra-wmi-mai-PS
P	NH-mai-avo-pks-sol-mai-ilm-sci-enr-OF
P	NH-mai-avo-pks-sol-mai-wmi-wen-hou-mai-avo-mer-wit-mai-NH
P	NH-mai-avo-wty-cor-syc-cor-bro-pks-sol-mai-PS
P	RL-mai-ilm-sci-enr-OF
P	NH-mai-col-sta-wit-mai-avo-gre-oak-rav-rad-maid-NH
P	NH-mai-hou-wen-wmi-gra-avo-mai-NH
P	NH-mai-wit-sta-nor-yal-avo-wit-nor-toor-har-wit-mai-NH
T	PS-mai-avo-pks-sol-mai-PS
P	PS-mai-avo-pks-sol-mai-PS
P	PS-mai-wmi-wen-hou-rax-gra-wmi-mai-PS
T	NH-mai-wit-sta-col-mai-NH
P	NH-mai-wit-sta-avo-gra-wmi-PS
P	PS-mai-avo-pks-sol-mai-PS
T	OH-brt-gre-gra-nct-nfl-vei-grm-cav-vei-nfl-nct-sar-hgh-har-lea-isl-far-wai-ken-cha-eve-nep-eve-cha-ken-cra-gra-mem-gre-wmi-PS
P	PS-wmi-wen-hou-mai-wit-aps-kld-how-mer-wit-mai-NH
P	PS-wmi-yal-avo-sta-col-mai-NH
P	PS-wmi-gra-mem-roy-avo-wmi-mai-PS
P	PS-wmi-wen-hou-mai-avo-mer-wit-mai-NH

Table 6.4. The list of test drives used in experiments. The first column denotes the use of the sequence: P – previous experience, T – test.

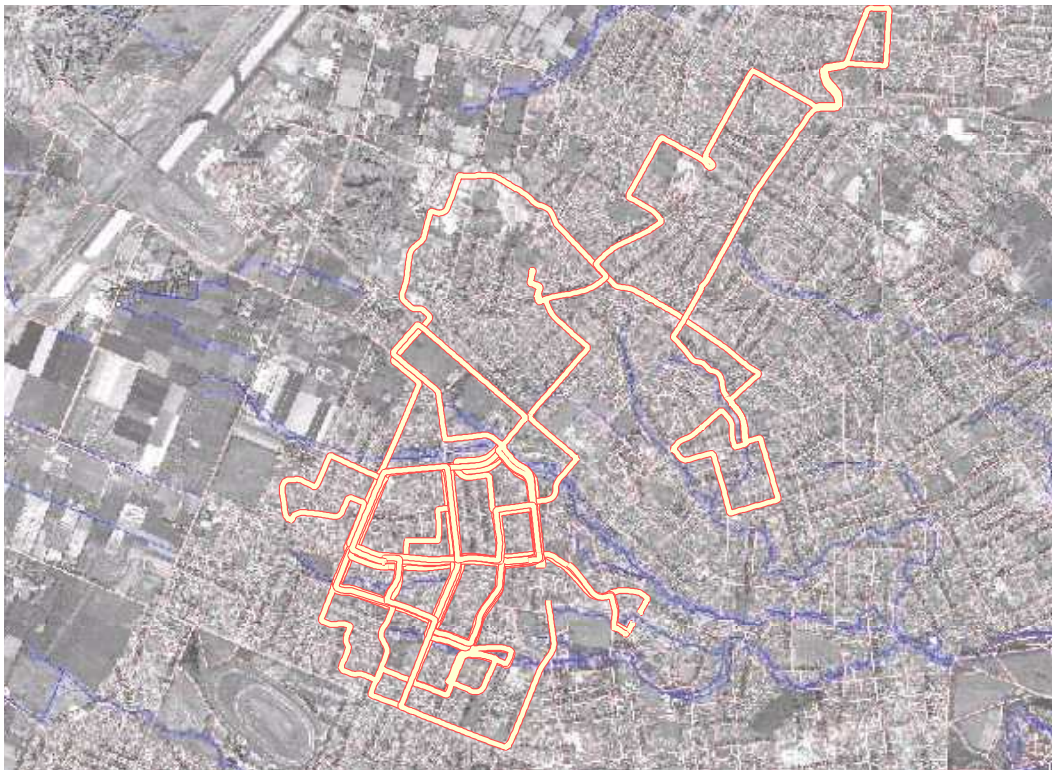

Figure 6.24. The map of all test drives presented over the aerophoto map of northwest Christchurch.

These test drives do not represent a random sample of someone's driving. Instead most of them are the same (or share parts of their routes). On the other hand, some drives (but not many of them) are very different. We believe that usual driving routes for a typical driver have similar characteristics. For testing we selected three drives. One of them has a "duplicate" drive with the same route among the drives selected for previous experience. The second drive has some similarity with previous experience drives, but after several events it changes its route and takes a new direction. The third drive is very different from all other routes.

To perform experiments with driving event prediction, we developed a set of Visual Basic for Applications macros in the same database where we stored the data about test drives and recognized events. For this experiment the data model of the database is further extended and presented in Figure 6.25.

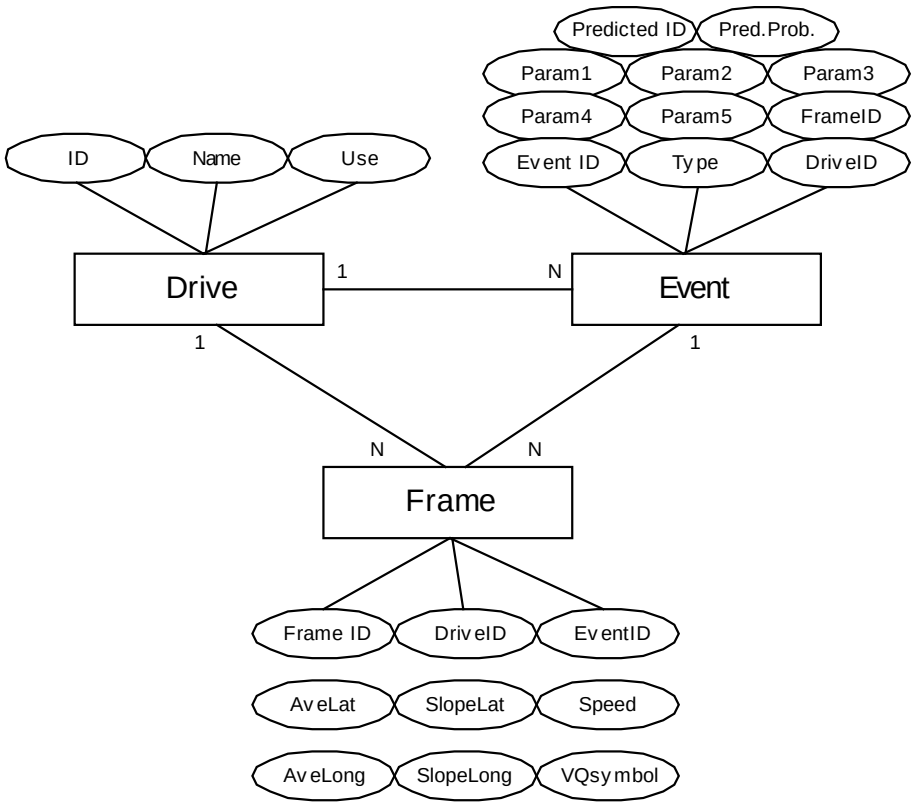

Figure 6.25. The data model of the database used to store data about drives and events, (presented by using simplified ER diagram).

The first macro generates Start/Stop events for all drives. The second macro iterates through all frames and recognizes drive-straight and short-break events. The next macro calculates event parameters for all events. Once these macros are executed the database is ready for testing. One macro creates the dynamic data structure for previous experience for all drives which we manually selected and marked in the database. Then, for each drive marked for testing, we execute the Predictor module. The Predictor module follows the procedure we have described above. It iterates through all events for a given drive and selects the most probable next event. This predicted event and the prediction probability are stored into an Events table for later analysis.

Firstly we used the data for recognized driving event together with numerical data for all test drives to build a complete sequence of driving events, with all parameters defined. The result of this process is again stored into the same database. Then we chose three sequences for testing, and the set of test drives which represents the previous driving experiences for this experiment. One test sequence is also part of the "previous experience" (PE) set, the second has a "twin" sequence (the same route driven) among the PE drives. There is no similar sequence in the PE set for the third drive.

The sequences from the PE set are loaded into the data structure, and the straight driving index is constructed during the process. Then we apply events from each of the test sequences. The predictor module generates hypotheses and calculates similarities between events, as well as the prediction probabilities for hypotheses. The graph of the prediction probabilities for each of these three test sequences are given in Figure 6.26, and numerical results are shown in Table 6.5. The program for this experiment is implemented in Visual Basic for Applications (VBA), and resides inside the same Microsoft Access database where drive data is originally stored.

st	rc	st	lc	lt	st	rc	st	lt	st	lc	rc	st	lt	lc	rc	st	lc	st	rt	st
.94	.87	.92	.81	.91	.84	.79	.94	.88	.90	.87	.92	.86	.91	.79	.80	.89	.86	.91	.86	.90
st	rc	st	lc	st	lt	st	lc	rc	st	lt	st	rc	st	lc	st	lt	st			
.86	.84	.90	.85	.92	.86	.88	.87	.70	.12	.76	.08	.03								
st	rc	st	lc	st	rc	st	lc	st	rc	st	…									
.74	.68	.24	.15	.06																

Table 6.5. Numerical values for event similarities for selected test drives.

As we can see in Figure 6.26., the first test drive, which has the same route as an another drive that is part of the previous experience, generates predictions with a high level of confidence. The system confidence increases as the drive continues without encountering unexpected events. Unexpected events are all events which have not been seen previously in a similar sequence of events. The result of increased confidence is a higher probability for the predicted event.

In the second test drive a few events at the beginning are on the same route as events in another drive that is part of the previous experience (see Table 6.4.). For these events, the similarity measure is very high and probability prediction is also high as the system slowly starts to gain confidence. However, the left turn comes too early and the similarity between two straight segments is very small, which causes the system to reduce its prediction probability and drop its confidence. The next two events are left turns and their similarity is reasonably high, so prediction probability rises again. However, the following straight segments have very different lengths and similarities are small, thereafter causing hypotheses similarity to drop below the threshold value and the system stops generating predictions.

The last test drive is very different from the others. The sequence of straight segments and right curve, as seen at the beginning of this drive, could be found among other drives, so some probability is generated. However, events after this are very different from what can be found among previous drives, and the prediction probability drops below the threshold value after just a few further events.

The results of tests presented in the table and the figures above show that our system for driving event prediction is able to reliably predict future driving events. Confidence in a predicted event (measured as a prediction probability) increases when longer sequences of repeated events are encountered over and over again. Such an example is presented with the first test drive above, and we expect this to be a common situation when the set of previously experienced drives becomes larger.

As previously discussed, our system generates a number of predictions (hypotheses) with various probabilities. However, due to a limited set of previous experiences in our test data set, the actual number of generated hypotheses was small and we decided not to evaluate the quality of predictions other than the best ones. We believe that a real-world system would have enough previous experiences to cover most

driving events in most common situations. Consequently, most driving events will be successfully predicted; with high probabilities.

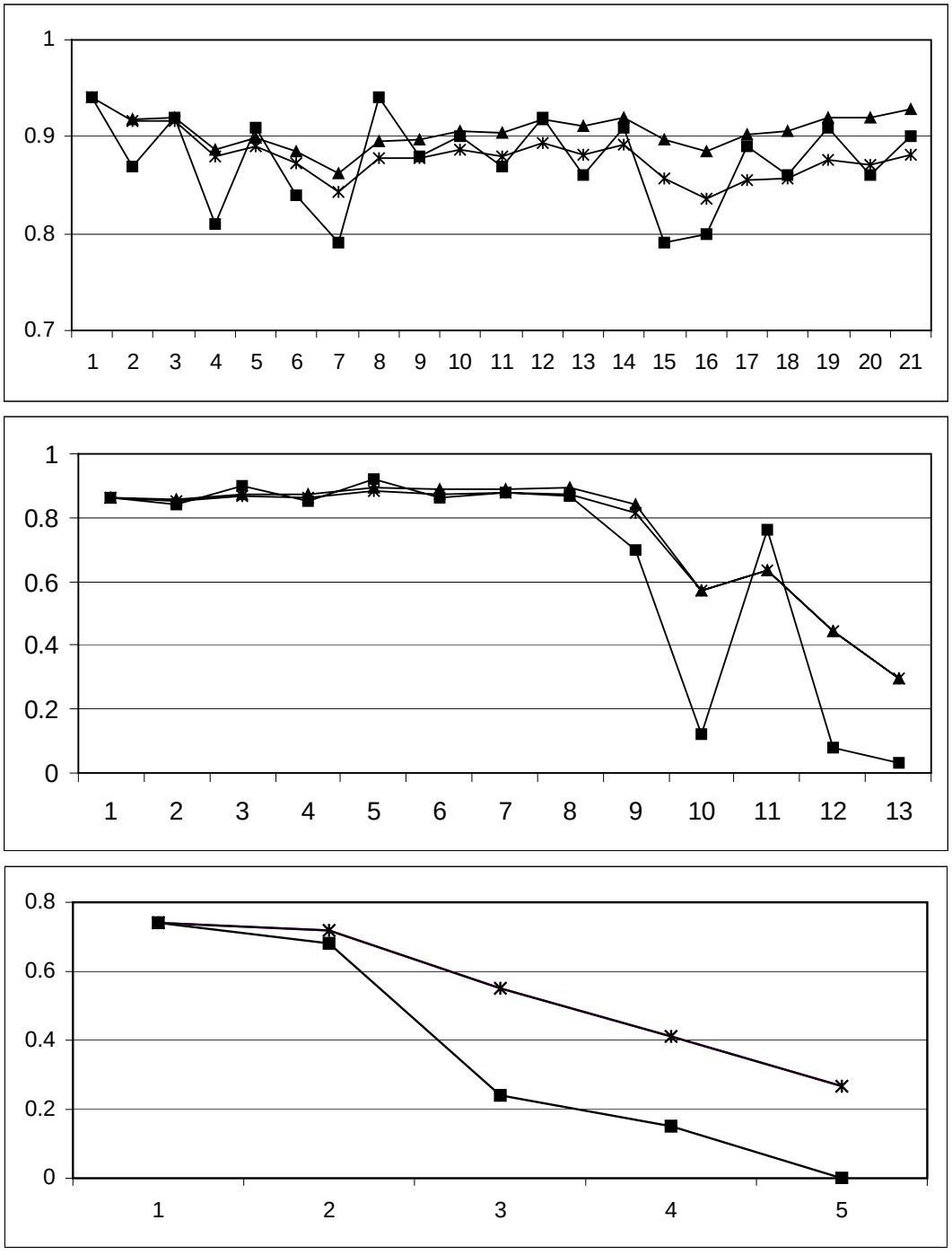

Figure 6.26. The calculated similarities of events (squares), hypotheses similarities from the base formula (stars) and prediction probabilities with added confidence factors (triangles) for selected test drives.

Situations where a driver experiences unexpected events would hopefully be rare. We may use the detected difference between the encountered and predicted events to indicate such problems. As the prediction probability for the predicted event is higher, and the similarity between events is lower, our confidence in the existence of problems will increase. Even a very simple combination of these factors, like $\delta = Z_{Ht-1} \bullet (Z_{Ht} - Z_{Ht-1})$ can be used to detect unexpected events. We will denote δ as the *difference measure*.

The following experiment will prove the detection capabilities of the proposed method. To overcome a limited number of test drives, we will use artificially generated sequences. We started from the first test drive described in the previous experiment. It is one of only two drives made along the same route. We copied it into twenty new sequences. Then we randomly changed parameters of these sequences to make them similar, yet different from the original. Ten of these sequences will represent a "normal" test case for driving along a previously experienced route (group 1). Similarities between artificial and real events in this group are the same as between events in two actual test drives along the same route (range [0.8-0.97]). The other ten sequences (group 2) represent significantly different driving experiences and similarities between events are randomly set to the range [0-1].

In both groups we changed the last four events to represent unexpected events. The first and third among the last four events in the actual drive are left curve and right turn respectively. The unexpected events in normal cases would be events of different types, or very different parameters. To accurately represent such events, we choose their similarities to be either zero or very small random value (< 0.1). The second and fourth events are driving straight events and, to preserve a logical sequence of events, we allowed these to be again events of the same type, but to have random similarity in the range [0-1]. Figures 6.27.a and 6.27.b present similarities between events in groups 1 and 2 respectively.

The actual driving sequence (from which we derived test sequences) is used as a sole previous experience and all hypotheses in these experiments are derived from it. For each artificial sequence, a hypothesis similarity is calculated using the method described above. Figure 6.27.c presents similarities for sequences from group 1. The similarity of each individual event is in the range [0.8-0.97]. However, the similarity of sequences steadily increases with the increase of confidence, as the stream of similar

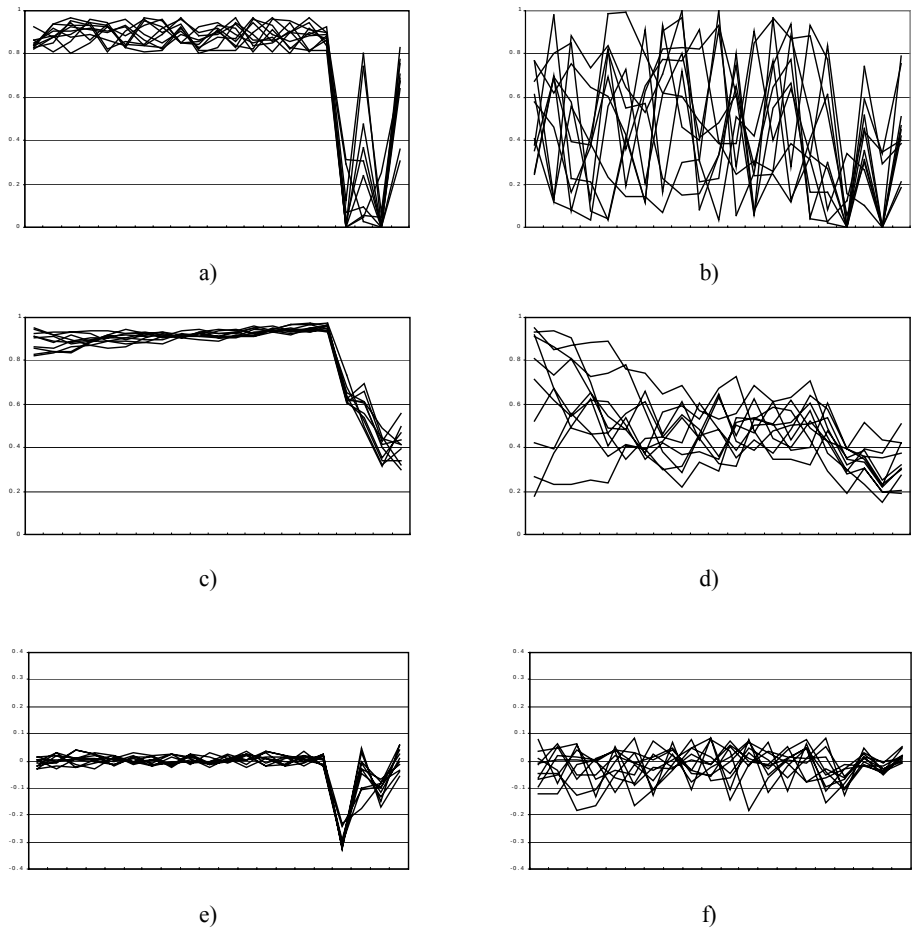

Figure 6.27. Detection of unexpected events.

events becomes longer. Near the end of each sequence we can be very confident that the hypothesis represents the same sequence of events as the test sequence and we may expect the next event to follow the same pattern. When an unexpected event occurs, the similarity quickly drops. The second group of sequences does not have a long chain of similar events and their similarity with a hypothesis does not achieve higher values, as can be seen in Figure 6.27.d.

As we proposed above, the combination of high confidence and decline in similarity may detect an unexpected event. Figures 6.27.e and 6.27.f present calculated values of difference measures for these two groups respectively. The values calculated when an unexpected event has appeared after a longer chain of similar events are much lower than when sequences are not very similar. This measurement can be used to detect potentially dangerous situations and in other applications as described in Chapter 3.

The method proposed here can be further improved by using more elaborate techniques if required. However, even a very simple set of formulas, as presented, proves that successful recognition of unexpected driving events is feasible.

6.8. Conclusions

The purpose of the experiments presented in this Chapter was to show that it is possible to reliably predict future driving events based on previous driving experiences as long as this is supported by a limited set of driving sensors and a reasonable amount of computing power. In experiments presented here we used symbolic machine learning techniques to overcome problems encountered with sub-symbolic techniques.

We use the term *driving event* to refer to any major change in vehicle position, attitude or speed. The ordered set of driving events while driving from one place to another is called a *route*. Here we first recognize driving event types from sensor data. Then we characterize events by measuring other important parameters. Driving events are further used to build a set of previous experiences. A current event is compared with previous similar events and one or more hypotheses are generated and evaluated. The future event is predicted from the hypothesis with the largest similarity to the currently observed set of events.

For driving event recognition we use hidden Markov models. Hidden Markov models are a popular method for modelling stochastic processes, whose internal state changes are not directly observable. Hidden Markov models have been used successfully in a number of different domains during the last decade. For driving event recognition we used discrete, left-to-right HMMs as they better capture the dynamic characteristics of data than a general HMM. Baum-Welch's re-estimation method was selected for training. We modified it to better handle numerical problems caused by the nature of our problem. We developed the WinHMM software for HMM training and evaluation. WinHMM has since also been used for event recognition in video streams.

Several processing steps were used to convert sensor data to symbols required for HMMs. Filtered and normalized data are segmented into 0.5 s frames by using a least squares approximation method. Vector quantization methods were used to convert five

continuous-domain frame parameters into symbols from the predefined set. We found that a codebook size of sixteen gives small quantization errors and good recognition performance for HMMs.

The seven most common driving event types, which could be found in New Zealand towns, were selected for these experiments. For each type we found the most appropriate number of states. A new model was created for each type and trained by a representative subset of events of the given type. We used data from 22 test drives and manually selected and labelled 238 events of the seven chosen types.

We implemented parallel evaluation of observation sequences representing driving events for all trained models. Each observation sequence is fed into the seven models, and the driving event type is determined from the model with the highest probability. We found events of left turns on intersections with and without roundabouts too similar to be distinguished. However, for all other cases, the system achieved a very high recognition rate of 98.3% (234 of 238). We also found that the average distance between the highest and the next largest probability was above 48%. This showed that the proposed method and the developed system are able to reliably recognize driving events using only a limited set of sensors.

Recognized driving events are then used to predict future driving events. Events are first defined by calculating other relevant parameters from the sensor data. Parameters that are used for event definition depend on event type. The similarity between two events of the same type is calculated by comparing their parameters. Similarities between the currently detected and previously encountered sequences of events are calculated by using an iterative reinforcement method. A previous sequence similarity is updated with a similarity measure for the currently detected event. When a long streak of similar events is encountered we use confidence factors to further increase similarity.

We developed a method and implemented a program for future event prediction. Previously experienced driving events are stored in a dynamic data structure, which is indexed for faster retrieval. Each newly recognized event is compared against previous experience data and new hypotheses are generated if necessary. Similarities between the current sequence of events and those in hypotheses are updated. Hypotheses are deleted or clustered, depending on their new state. The most probable driving event is chosen

from the hypothesis with the highest similarity. Our test case, though relatively small, demonstrated successful prediction of driving events.

We also demonstrated that the proposed method can successfully detect unexpected events. The combination of a high confidence in the current situation and the decline in similarity due to different nature of the event can result in a decline in the difference measure which is much larger than in cases when confidence is not as high. As the set of previous experiences grows and becomes similar to the experiences of a real driver, the system will be able to provide successful prediction in a large majority of circumstances. In such situations, the system will always make hypotheses with a large confidence. Any unexpected events which appear in such situations will be easily identified by the system and a warning for the driver can be generated.

7. Conclusions

It is human nature to travel and explore new environments. Transportation is one of the major driving factors for economic development. Automobiles are certainly the most popular and most commonly used vehicles. Vehicle driving is an act of operating and directing the course of a vehicle. Driving is a complex decision-making process in a multidimensional, highly dynamic environment. Errors in making driving decisions are generally rare, but often with serious consequences. They cause large human and financial losses every year.

Experience is a significant source of knowledge for any human activity. Knowledge about past failures may help to avoid similar failures in the future, while repeating or even improving successes. Because driving is a complex and dynamic activity, extensive previous experience can be of enormous importance when important decisions need to be made quickly.

In this thesis we have explored a number of ways for using machine learning techniques to learn driving patterns from previous experience and to exploit learned patterns to support the driver in fulfilling navigation tasks. Here we summarize the main contributions of this thesis.

7.1. Framework of the Driving Warning System

Driving patterns are sequences of events in the traffic system, which are repeated over time. They occur often due to regularities in the traffic infrastructure and a number of other reasons. We postulate that by identifying and learning patterns in someone's driving, a computer-based system can provide information that can help the driver to

fulfil a range of different navigational driving tasks. This could be done by comparing predicted and actual driving events which may detect previously unseen situations.

Situation awareness is a task-specific understanding of the current situation including position of the driven vehicle, other vehicles in the vicinity, and intentions of their drivers. Recent research explicitly declared the need for situation awareness in intelligent transportation systems. We proposed a model for using methods for learning driving patterns as building blocks for constructing the situation model.

Here we proposed a framework of the driver warning system based on learning driving patterns. The goal of the framework is to warn the driver if/when his/hers current behaviour becomes significantly different from usual, in the context of the routes he/she usually travels. The learning part of the system uses input from navigation sensors to build and maintain the traffic system model. This model consists of the current state of the system, recent history of the system, and the probability that the model represents the actual state of the traffic system. The prediction module finds patterns from the pattern history that are similar to the pattern in the current traffic system model. This set of similar patterns is used by the prediction module to calculate the most likely future events. If the difference between the actual and predicted states is significant, then the specific pattern of events signifies either a new experience or some uncommon situation, which may cause an accident. We believe that a number of new experiences for a typical driver would be very small, so we can generate a warning for the driver from a detected difference.

7.2. Data Acquisition System

To collect data from navigation sensors required for our experiments in learning driving patterns we needed a data acquisition system. Such data acquisition systems are routinely built and commercially available. However, we had a very small budget for the project, and had to implement the data acquisition system by using an unorthodox design, of-the-shelf hardware and software components, and a familiar software development environment. In this respect, the developed data acquisition system represents one of the contributions of this thesis.

The data acquisition hardware is designed around a standard desktop PC, powered by the car battery through standard power inverter unit. The computer is equipped with the keyboard and speakers to enable basic user interaction. To collect data, we used a GPS receiver and a pair of accelerometers connected to the analogue-to-digital conversion card. The data acquisition software was designed and implemented using object-oriented techniques and tools. Collected data from accelerometers are digitally processed to increase the signal-to-noise level. A simple averaging technique is used while reading data from the conversion card, and a second order low-pass Butterworth digital filter.

The data acquisition system was successfully used to collect a large amount of data for our experiments with driving pattern learning. This system was also used by a postgraduate student in the Psychology Department at the University of Canterbury.

7.3. Vehicle Motion Prediction by Neural Networks

Neural networks are computational models developed to mimic human information processing. A time-series prediction has quickly become a popular application of neural networks, as their prediction capabilities exceeded these of conventional methods in many domains.

The analysis of neural network capabilities for predicting a vehicle motion has been conducted through a number of experiments in this project. We conducted experiments to explore the possibilities of short-term prediction of vehicle movement by neural networks and to compare prediction performances of various neural network architectures. We predicted future values of lateral and longitudinal acceleration based on current measurements of these values (obtained from the data acquisition system).

In the first set of experiments, we used multi–layer perceptrons. The results suggested that the prediction capabilities for short time in advance are very good and that predicted accelerations are very close to actual ones. However, the prediction error increases substantially for larger prediction orders, and networks are not able to correctly predict accelerations for driving events such as turns. In the second set of experiments we compared the prediction capabilities of various neural network

architectures for a short period of time in advance. We found that there is no major difference in the results among architectures and that prediction capabilities are reasonably good. Multi-layer perceptrons performed comparably with more complex architectures and required shorter times for training.

The third set of experiments explored longer–term predictions with various architectures. This time, time–delay and recurrent networks, which are designed for time-series prediction tasks, performed better than multi–layer perceptrons. However, predicted accelerations were still very different from the actual values and the required processing power for training becomes very high. We concluded that symbolic machine learning techniques should be used for longer term prediction instead of neural networks.

7.4. Driving Event Recognition and Prediction

We use the term driving event to refer to any major change in vehicle position, attitude or speed. We proposed a method for driving event recognition based on discrete, left-to-right hidden Markov models, and using data from accelerometers and speed measured by GPS receiver. Sensor data are converted to symbols required for discrete HMM through several processing steps.

Hidden Markov models are a popular method for modelling stochastic processes, whose internal state changes are not directly observable. For experiments, we developed the WinHMM software for HMM training and evaluation. WinHMM has since also been used for event recognition in video streams at University of Twente, Netherlands.

Seven most common driving event types, which could be found in New Zealand towns, were recognized in our experiments. A new model was created for each driving event type and trained by a representative subset of events of the given type. We used data from 22 test drives and manually selected and labelled 238 events of the seven chosen types.

We implemented parallel evaluation of observation sequences representing driving events for all trained models. Each observation sequence is fed into all seven models, and the driving event type is determined from the model with the highest probability.

We found events of left turns on intersections with and without roundabouts too similar to be distinguished. However, for all other cases, the system achieved a very high recognition rate of 98.3% (234 of 238) and a model with the highest probability is clearly distinguished. This showed that the proposed method and the developed system are able to reliably recognize driving events using only a limited set of sensors.

We developed a method and implemented a program for future event prediction. Previously experienced driving events are stored in a dynamic data structure, which is indexed for faster retrieval. Each newly recognized event is compared against previous experience data and new hypotheses are generated if necessary. Similarities between the current sequence of events and those in hypotheses are calculated by using an iterative reinforcement method. When a long streak of similar events is encountered, we use confidence factors to further increase similarity. Hypotheses can be clustered or removed, depending on their new state. The most probable driving event is chosen from the hypothesis with the highest similarity. Our test case, though relatively small, demonstrated successful prediction of driving events, fast reaction of the system to unexpected situations, and the possibility of recovery. As the set of previous experiences grows and becomes similar to the experiences of a real driver, the system will provide successful prediction in a large majority of circumstances. Any unexpected events will be clearly identified by the system.

7.5. Concluding Remarks

Research in the area of Intelligent Transportation System was almost exclusively conducted by large research teams supported by the large grants from major automobile producers. Working on large, well-resourced research projects has advantages, but very often limits the creativity of the people involved as expectations are also very large. We believe that this thesis proved that comparable research is possible even on a very limited budget, and we hope that it may serve as inspiration for other researchers, particularly from small countries.

The research presented in this thesis could be extended in a number of ways. Our results in predicting future driving events look very promising and it could be further evaluated by collecting a much larger set of data to be used as previous experiences.

This would also require minor changes to the system as outlined with the description of the system.

Further verification of the proposed methodology could be achieved by modifying the system to operate in the real-time environment. By using a live system, the usability of generated warnings could be measured. For some purposes, warnings generated by the proposed system could be too late. However, the response time of the system could be increased by operating on lower-level driving events, like gear changes.

Another interesting avenue for further research would be to integrate short-term and long-term predictions into a composite system. Such a system may use generated long-term predictions to create a context in which short-term predictions would be made. This may improve the quality of the short-term predictions, and provide a basis for confidence estimation required for warning generation. Such a method may ensure the short response time with a reasonable prediction performance and a reliable warning generation.

8. References

[ADLink, 1997] ADLink Technology Inc, *Manual for ACL-8112HG/DG ver C Enhanced Multi-function Data Acquisition Card*, second edition, 1997.

[An, 1996] An, P.E., and Harris, C.J., An intelligent driver warning system for vehicle collision avoidance. *IEEE Trans. System, Man and Cybernetics,* Vol. 26, No. 2, pp. 254-261, 1996.

[Analog, 1996] Analog Devices, *ADXL05 – Single chip Accelerometer with Signal Conditioning*, 1996.

[Ashbrook, 2002] Ashbrook D., Starner T., Learning Significant Locations and Predicting User Movement with GPS, In *Proceedings of the 6th IEEE International Symposium on Wearable Computers*, Seattle WA, pp. 101-108, 2002.

[Asimov, 1950] Asimov I., *I, Robot*, Ballantine Books, New York, NY, 1950

[Austin, 1996] Austin T., DiGenova F., Carlson T., Joy R.W., Gianolini K.A., Lee J.M., *Characterization of Driving Patterns and Emissions from Light-Duty Vehicles in California*, Research Note 96-11, California Environmental Protection Agency, Air Resources Board, 1996.

[Aycard, 1997] Aycard O., Charpillet F., Foht D., Mari J.F., Place Learning and Recognition using Hidden Markov Models, *In Proceedings of IEEE IROS97 conference*, pp 1741-1746. 1997.

[Baker, 1975] Baker, J. K., The dragon system - An overview, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 23, No. 1, pp 24–29, 1975.

[Baum, 1966] Baum L.E., Petrie T., Statistical interference for probabilistic functions of finate state Markov chains, *Annals of Mathematical. Statistics*, Vol. 37, pp. 1554-1563, 1966.

[Baum, 1967] Baum L.E., Egon J.A., An inequality with applications to statistical estimation for probabilistic functions of a Markov process and to a model for ecology, *Bull. Amer. Meteorol. Soc*., Vol. 73, pp. 360-363, 1967.

[Baum, 1968] Baum L.E., Sell G.R., Growth functions for transformations on manifolds, *Pac. J. Math*, Vol. 27, No 2, pp 211-227, 1968.

[Baum, 1970] Baum L.E., Petrie T., Soules G., Weiss N., A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains, *Annals of Mathematical Statistics*, Vol. 41, No. 1, pp. 164-171, 1970.

[Baum, 1972] Baum L.E., An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes, *Inequalities*, Vol. 3, pp. 1-8, 1972.

[Bengio, 1994] Bengio Y., Simard P., Frasconi P., Learning Long-Term Dependencies with Gradient Descent is Difficult, *IEEE Transactions on Neural Networks*, Vol. 5, No. 2, pp157-166, 1994.

[Bengtsson, 2001] Bengtsson J., *Adaptive Cruise Control and Driver Modeling*, PhD thesis, Lund Institute of Technology, Sweden, 2001.

[Bobrow, 1968] Bobrow D. G., Natural Language Input for a Computer Problem-solving System, In M. Minsky (eds), *Semantic Information Processing*, MIT Press, Cambridge, MA, 1968.

[Bonasso, 91] Bonasso R. P., Integrating Reaction Plans and Layered Competences Through Synchronous Control, In Proceedings of the Internation Joint Conference on Artificial Intelligence (IJCAI), Sydney, pp. 1225-1231. 1991.

[Bowerman, 1993] Bowerman B.L., O'Connell R.T., *Forecasting and time series: an applied approach*, Duxbury Press, Belmont, CA.,1993.

[Brooks, 1986] Brooks R.A., A Robust Layered Control System for a Mobile Robot, *IEEE Transaction on Robotics and Automation*, Vol. RA2, No. 1, pp. 14-23, 1986.

[Buchanan, 1978] Buchanan B., Feigenbaum E., DENDRAL and Meta-DENDRAL: Their applications dimension, *Artificial Intelligence,* Vol. 11, No. 1, pp 5-24, 1978.

[Capek, 1921] Capek K., *R.U.R.: Rossum's Universal Robots*, Prague National Theatre, 1921.

[Catling, 1990] Catling I., Prometheus and Drive-European Initiatives, in Walker J., (ed.), *Mobile Information Systems*, Artech House, Boston, MA,1990.

[Cajal, 1911] Ramon y Cajal S., *Histologie du systeme nerveux de l'homme et des vertebres*, Paris, 1911.

[Chen, 1990] Chen, S., Billings, S.A., Grant P., Nonlinear system identification using neural networks, *International Journal of Control*, Vol 51., No 6, pp. 1191-1214, 1990.

[Coad, 1991] Coad P., Yourdon E., *Object Oriented Design*, Yourdon Press Computing Series, Englewood Cliffs, NJ, 1991.

[Connell, 1990] Connell J., *Minimalist Mobile Robotics: A Colony-Style Architecture for an Artificial Creature*, Academic Press, Boston, MA., 1990.

[Connell, 1991] Connell J., SSS: A HybridArchitecture Applied to Robot Navigation, In *Proceedings of the IEEE Conference on Robotics and Automation*, 1992.

[Dai, 1995] Dai J., Robust Estimation of HMM Parameters Using Fuzzy Vector Quantization and Parzen's Window, *Pattern Recognition*, Vol. 28, No. 1, pp. 53-57, 1995.

[Darragh, 1990] Darragh J.J, Witten I.H., James M.L., The Reactive Keyboard: A Predictive Typing Aid, *IEEE Computer*, Vol. 23., No. 11., pp. 41-49. 1990.

[Deller, 1993] Deller J.R., Proakis J.G., Hansen H.L., *Discrete-time processing of speech signals*, New York: Macmillan, 1993.

[EI-Gindy, 1993] EI-Gindy M., Palkovics L., Possible application of artificial neural networks to vehicle dynamics and control: a literature review, *International Journal of Vehicle Design*, Vol. 14, No 5-6, pp. 592-614, 1993.

[Elman, 1990] Elman L., Finding Structure in Time, *Cognitive Science*, Vol. 14, No. 2, pp. 179-211, 1990.

[Engle, 1982] Engle, R.F.,. Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. *Econometrica* Vol. 50, pp. 987-1007, 1982.

[Ericsson, 2001] Ericsson E., Independent driving pattern factors and their influence on fuel use and exhaust emission factors. *Transportation Research*, Part D., Vol. 6, pp. 325-345, 2001.

[Evans, 1968] Evans T.G., A program for the solution of geometric analogy intelligence test questions. In M. Minsky (eds), *Semantic Information Processing*, MIT Press, Cambridge, MA, 1968.

[Forney, 1973] Forney G.D., The Viterbi Algorthm, *Proceedings of the IEEE*, Vol. 6, No. 3, pp. 268-278, 1973.

[Fraile, 1998] Fraile R., Maybank S.J., Vehicle Trajectory Approximation and Clssification, *In Proceedings of the British Machine Vision Conference*, pp 832-840, 1998.

[Frank, 1983] Frank R.L., Current developments in Loran-C, *Proc. IEEE*, Vol. 71, No. 10, pp 1127-1139, 1983.

[Friedberg, 1958] Friedberg R M., A Learning Machine - Part I, *IBM Journal of Research and Development*, Vol. 2, No. 1, pp. 2-11, 1958.

[Gershenfeld, 1993] Gershenfeld N.A., Weigend. A.S., The future of time series: Learning and understanding, In Weigend A. S., and Gershenfeld N. A., editors, *Time Series Prediction: Forecasting the Future and Understanding the Past*, Addison-Wesley, Reading, Mass., pp. 1-70, 1993.

[Ghazizadeh, 1996] Ghazizadeh A., Fahim A., EI-Gindy M., Neural network representation of a vehicle model: Neuro Vehicle, *International Journal of Vehicle Design*, Vol. 17, :No. 1, pp. 55-75, 1996.

[Ghazizadeh, 1997] Ghazizadeh A., Fahim A., EI-Gindy M., Neural network and fuzzy logic applications to vehicle systems, *International Journal of Vehicle Design*, Vol. 18, :No. 2, pp. 132-193, 1997.

[Gibbs, 1997] Gibbs W.W., Transportation's Perennial Problems, *Scientific American*, No. 10, pp. 32-35, 1997.

[Gori, 1989] Gori M., Bengio Y., De Mori R., BPS: A learning algorithm for capturing the dynamical nature of speech. In *Proceedings of the International Joint Conference on Neural Networks*, pp. 643--644 Washington D.C. 1989.

[Grasso, 1998] Grasso R., Prévost P., Ivanenko, Y. & Berthoz A.,. Eye-head coordination for the steering of locomotion in humans: an anticipatory synergy, *Neuroscience Letters*, 253, pp. 115-118, 1998.

[Gray, 1984] Gray R. M., Vector Quantization, *IEEE Acoustics, Speech and Signal Processing Magazine*, Vol. 1, No. 2, pp. 4-29, 1984.

[Greenblat, 1967] Greenblat R.D., Eastlake D.E., Crocker S.D., The Greenblat Chess Program. In *Proceedings of the Fall Joint Computing Conference*, pp 801-810, San Fransisco, 1967.

[Griswold, 1994] Griswold N.C., Kchtarnavaz N.D., Miller K.M., Transportable neural network controller for autonomous vehicle following, In *Proceedings of Intelligent Vehicles Symposium*, pp. 195-200, 1994.

[Hartley, 1958] Hartley H., Maximum likelihood estimation from incomplete data, *Biometrics*, Vol. 14, pp. 174–194, 1958.

[Hartley, 1991] Hartley P., Pipitone F., Experiments with the Subsumption Architecture. In *Proceedings of the International Conference on Robotics and Automation*, pp. 1652-1658, 1991.

[Haskell, 2000] Haskell R. E., Hanna D. M., Li P., Cheok K. C. Hudas G., Finding Pattern Behavior in Temporal Data using Fuzzy Clustering, In *Proceedings of the Conference on Artificial Neural Networks in Engineering*, pp. 703 – 711, 2000.

[Haykin, 1994] Haykin S., *Neural Networks, A Comprehensive Foundation*, Prentice Hall, Englewood Cliffs, NJ, 1994.

[Hofmann, 1997] Hofmann-Wellenhof B., Lichtenegger H., Collins J., *Global positioning system: theory and practice*, Springer-Verlag, Wien, New York, 1997.

[Horne, 1995] Horne, B.G., Giles, C.L., An experimental comparison of recurrent neural networks, In Tesauro, G., Touretzky, D., Leen, T., editors, *Advances in neural Information processing Systems 7*, MIT Press, , Cambridge, MA, pp. 697-704, 1995.

[Hornik, 1991] Hornik K., Approximation Capabilities of Multilayer Feedforward Networks, *Neural Networks*, Vol. 4, No. 2, pp 251-257, 1991.

[Horowitz, 1975] Horowitz S.L., A Syntactic Algorithm for Peak Detection in Waveforms with Applications to Cardiography, *Communications of the ACM*, Vol. 18, No. 5, pp. 281-285, 1975.

[Hsu, 2002] Hsu F., Behind Deep Blue: Building the Computer that Defeated the World Chess Champion, Princeton University Press, Princeton N.J., 2002

[HTK, 2002] HTK History web page available at:
http://htk.eng.cam.ac.uk/docs/history.shtml, last update: December 2002.

[Hu, 1964] Hu M.J.C., *Application of the adaline system to weather forecasting*, Master thesis, Technical report 6775-1, Stanford Electronic Laboratories, Stanford, 1964.

[Jelinek, 1975] Jelinek F., Bahl L., Mercer R., Design of a linguistic statistical decoder for the recognition of continuous speech, *IEEE Transactions on Information Theory*, Vol. 21, No. 3, pp. 250–256, 1975.

[Jochem, 1996] Jochem, T., Pomerleau, D., Life in the Fast Line: The Evolution of an Adaptive Vehicle Control System, *AI magazine*, Vol. 17, No. 2, pp. 11-50, 1996.

[Jordan, 1986] Jordan M.I., Attractor dynamics and parallelism in a connectionist sequential machine, In *Proceedings of the Eighth Annual conference of the Cognitive Science Society*, pp 531-546. 1986.

[Kanungo, 2000] Dr. Tapas Kanungo web page available at:

http://www.cfar.umd.edu/~kanungo, last update: 8 May 2000.

[Kitchin, 1995] Kitchin C., *Using accelerometers in Low g Applications*, Analog Devices Application Note AN-374, 1995.

[Knowling, 1999] Knowling M., CARiN Navigation - The Latest and Greatest in In-Car Technology, *AutoSpeed*, Issue 43, 1999.

[Kohonen, 1982] Kohonen T., *Self-organized formation of topologically correct feature maps*, Master's thesis, Technische Universit at Munchen, also: *Biological Cybernetics*, Vol. 43, No. 1, pp. 59—69, 1982.

[Kraiss, 1990] Kraiss K.F., Kuttelwesch H., Teaching neural networks to guide a vehicle through an obstacle course by emulating a human teacher, in *Proceedings of International Joint Conference on Neural Networks*, pp. 333-337, 1990.

[Kraiss, 1993] Kraiss K.P., Kuttelwesch H., Identification and application of neural operator models in a car driving situation', in *Proceedings of IFAC Symposia Series n 5*, pp. 121-126, 1993.

[Kremer, 2001] Kremer C.S., Spatio-temporal Connectionist Networks: A Taxonomy and Review, *Neural Computation* Vol. 13, No. 2, pp. 249-306, 2001.

[Krogh, 1994] Krogh A., Mian I. S., Haussler D., A hidden Markov model that finds genes in E. coli DNA, *Nucleic Acids Research*, Vol. 22, pp. 4768-4778, 1994.

[Kuge, 2000] Kuge N., Yamamura T., Shimoyama O., Liu A., A Driver Behavior Recognition Method Based on a Drive Model Framework, In *Proceedings of the SAE World Congress*, 2000.

[Lamm, 1999] Lamm R., Psarianos R., Mailaender T., *Highway design and traffic safety engineering handbook,*. McGraw-Hill, New York, NY, 1999.

[Lang, 1990] Lang, K., Waibel, A. and Hinton, G. E., A time-delay neural network architecture for isolated word recognition. *Neural Networks*, Vol. 3, No. 3, pp. 23-43. 1990.

[Lapedes, 1987] Lapedes A., Farber R., *Nonlinear signal processing using neural networks: Prediction and system modelling*, Tech. Rep. LA--UR--87--2662, Los Alamos National Laboratory, Los Alamos, 1987.

[Lee, 1984] Lee K.C., Blanchard G.W., Rosser M.S., *Driving patterns of private vehicles in New Zealand*, Report to the Liquid Fuels Trust Board, New Zealand, 1984.

[Lee, 1990] Lee K.F., Hon H.W., Reddy R., An Overview of the SPHINX Speech Recognition System, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 38, No. 1, pp. 35-45, 1990.

[Legat, 2001] Legat K., Lechner W., Integrated navigation for pedestrians, In *Proceedings of the 5th GNSS International Symposium*, Seville, 2001.

[Lin, 1996] Lin T., Horne B., Tino P., Giles C.L., Learning Long-Term Dependencies is not as Difficult with NARX Recurrent Neural Networks, In Touretzky, D., Mozer, M., and Hasselmo, M., editors, *Advances in Neural Information Processing Systems 8*, pp. 577, MIT Press, Cambridge, MA. 1996.

[Liu, 1997] Liu A., Pentland A., Towards real-time recognition of driver intentions. In *Proceedings of the IEEE Intelligent Transportation Systems Conference*, pp 236-241, 1997.

[LTSA, 2004] Land Transport Safety Authority New Zealand, Accident statistics available at: http://www.ltsa.govt.nz/research/ , last updated 07/09/2004.

[Makhoul, 1985] Makhoul J., Roucos S., Gish H., Vector Quantization in Speech Coding, *Proceedings of the IEEE*, Vol. 73, No. 11, pp. 1551-1567, 1985.

[McCulloch, 1943] McCulloch, W.S., Pitts, W, A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, Vol. 5. pp 115-133, 1943.

[McDermott, 1982] McDermott, J., R1: A Rule-Based Configurer of Computer Systems, *Artificial Intelligence*, Vol.. 19, No. 1, pp. 39-88, 1982.

[Minsky, 1969] Minsky, M., Papert, S., *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, MA. 1969.

[Mitchell, 1977] Mitchell T.M., Version Spaces: A Candidate Elimination Approach to Rule Learning. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pp 305-310, 1977.

[Mitrovic, 1995] Mitrovic D., Murugesh S., Djordjevic S., Stanic Z., Possible Causes of Errors and Ways to Correct Them in the Global Positioning System, In *Proceedings of the TELSIKS conference*, pp 424-427, 1995.

[Moody, 1998] Moody J., Forecasting the Economy with Neural Nets: A Survay of Chellenges and Solutions, in Orr G.B., Muller K.R., (editors) *Neural Networks: Tricks of the Trade*, Springer, 1998.

[Mozer, 1993] Mozer, M.C., Neural net architectures for temporal sequence processing, In Weigend A. S., and Gershenfeld N. A., editors, *Time Series Prediction: Forecasting the Future and Understanding the Past*, Addison-Wesley, Reading, Mass.,, pp. 243-264, 1993.

[Narendra, 1990] Narendra K.S., Parthasarathy K., Identification and Control of dynamical Systems Using Neural Networks, *IEEE Transactions of Neural Networks*, Vol. 1, No. 2, pp. 4-27, 1990.

[NASA, 1997] NASA sojourner web page available at: http://mars.jpl.nasa.gov/MPF/rover/sojourner.html, last update: 8 July 1997.

[Nath, 1998] Nath N.M., CAN protocol eases automotive-electronics networking. *EDN*, August 1998.

[Nechyba, 1998] Nechyba M. C., Xu Y., Stochastic Similarity for Validating Human Control Strategy Models, *IEEE Transactions of Robotics and Automation*, Vol. 14, No. 3, pp. 437-451, 1998.

[Nefian, 1998] Nefian A.V., Hayes M.H., Hidden Markov models for face recognition, In *Proceedings of the ICASSP98*, pp. 2721--2724, 1998.

[Neuro, 2001] NeuroSolutions version 3. web page available at: http://www.nd.com/products/nsv3.htm, last update2001.

[Nilsson, 1984] Nilsson N.J., *Shakey the robot*, Technical Report 223, SRI International, 1984.

[Nourbakhsh, 1993] Nourbakhsh I., Morse S., Becker C., Balabanovic M., Gat E., Simmons R., Goodridge S., Potlapalli H., Hinkle D., Jung K., Van Vactor D., The Winning Robots from the 1993 Robot Competition, *AI Magazine*, Vol. 14, No. 4, pp. 51-62, 1993.

[Oliver, 2000] Oliver N., Pentland A., Graphical Models for Driver Behavior Recognition in a SmartCar, *In Proceedings of the IEEE Intl. Conference on Intelligent Vehicles*, 2000.

[Onken, 1994] Onken R., DAISY, an Adaptive, Knowledge-based Driver Monitoring and Warning System, In *Proceedings of the IEEE Vehicle Navigation and Information System Conference*, pp. 3-10, 1994.

[OnStar, 2003] www.OnStar.com

[Owen, 1996] Owen C., Nehmzow U., "Route Learning in Mobile Robots through Self-Organisation", In *Proceedings of the Eurobot 96*, 1996.

[Ozaki, 1982] Ozaki T., The statistical analysis of perturbed limit cycle processes using nonlinear time series models, *Journal of Time Series Analysis*, Vol. 3, pp. 29-41, 1982.

[Parkinson, 1996] Parkinson B.W., Spilker J.J., eds., *Global positioning system: theory and applications*, American Institute of Aeronautics and Astronautics, Washington, DC, 1996.

[Pavlidis 1973] Pavlidis T., Waveform Segmentation Through Functional Approximation, IEEE Transactions on Computers, Vol. 22, No. 7, pp. 689-697.

[Pentland, 1995] Pentland A., Liu A., Towards augmented control systems, In *Proceedings of the IEEE Intelligent Vehicles*, pp. 350-355, 1995.

[Pentland, 1999] Pentland A., Liu A., Modeling and prediction of human behavior, *Neural Computation*,Vol. 11, No. 1, pp. 229-242, 1999.

[Petkovic, 1997] Petkovic M., *Object-Oriented, Intelligent System for Vehicle Navigation*, Master Thesis, University of Nis, Yugoslavia, 1997.

[Petkovic, 2003] Petkovic M., Jonker W., *Content-Based Video Retrieval, A Database Perspective*, Kluwer Academic Publishers, Boston, MA, 2003.

[Pomerlau, 1989] Pomerlau D. A., ALVINN: An Autonomous Land Vehicle In a Neural Network, In D.S.Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan Kaufmann, 1989.

[Pomerleau, 1993] Pomerleau, D., *Neural Network Perception for Mobile Robot Guidance*, Kluwer Academic Publishers, Boston, 1993.

[Principe, 2000] Principe J.C., Euliano N.R., Lefebvre W.C., *Neural and Adaptive Systems, Fundamentals through Simulations*, John Wiley & Sons, 2000.

[Quinlan 1979] Quinlan J. R., Discovering Rules by Induction from Large Collections of Examples, In D. Michie (eds.), *Expert Systems in the Micro-Electronic Age*, Edinburgh University Press, Edinburgh, 1979.

[Rabiner, 1989] Rabiner L.R., A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, *Proceedings of the IEEE*, Vol. 77, No. 2, pp. 257-286, 1989.

[RoboCup, 2003] The RoboCup Federation web page, available at: http://www.robocup.org/, last updated 9. September 2003.

[Rombaut, 1993] Rombaut M., ProLab 2 : a driving assistance system, Proc of the *IEEE/Tsukuba International Workshop on Advance Robotics*, pp. 97-101, 1993.

[Rosen, 1970] Rosen D., Mammano F., Favout R., An Electronic Route-Guidance System for Highway Vehicles, *IEEE Transactions on Vehicular Technology*, Vol. 19, No. l, pp. 143-152, 1970.

[Rosenblatt, 1958] Rosenblatt, F., The Perceptron: a Probabibilistic Model for Information Storage and Organization in the Brain, *Psychological Review*, Vol. 65, pp.386-408. 1958.

[Rosenblatt, 1997] Rosenblatt, J. DAMN: A Distributed Architecture for Mobile Navigation. In Hebert M., Thorpe C., Stentz T., (editors), *Intelligent Unmanned Ground Vehicles: Autonomous Navigation Research at Carnegie Mellon*, Kluwer Academic Publishers, Norwell, MA, 1997.

[Rudd, 1993] Rudd K.E., Maps, genes, sequences, and computers: an Escherichia coli case study. *ASM News*, Vol. 59, pp. 335–341, 1993..

[Rumelhart, 1986] Rumelhart D.E., Hinton G.E., Williams R.J., Learning internal representations by error propagation, In Rumelhart D.E., McClelland, J. L., (editors), *Parallel Distributed Processing: Explorations in the*

Microstructure of Cognition, Volume 1, MIT Press, Cambridge, MA, pp. 318-362, 1986.

[Sage, 1983] Sage G. F., Luse J. D., Integration of Transit, Omega and Loran-C for marine navigation, *Journal of The Institute of Navigation*, Vol. 30, No. 1-4, pp. 22-33, 1983.

[Scheinman, 1987] Scheinman V., Robotworld: A multiple robot vision guided assembly system. In Proceedings of the 4th International Symposium on Robotics Research, 1987.

[Sejnovski, 1986] Sejnovski, T.J., Rosenberg, C.R., *NETtalk: a Parallel Network that Learns to Read Aloud*, Technical Report JHU/EECS-86/01, John Hopkins University, 1986.

[Siegelmann, 1997] Siegelmann H., Horne B., Giles C., Computational Capabilities of Recurrent NARX Neural Networks, *IEEE transaction on Systems Man and cybernetics – Part B: Cybernetics*, Vol. 27, No 2, pp. 208-215, 1997.

[Simon, 1958] Simon H. A., Newell A.,. Heuristic Problem Solving: The Next Advance In Operations Research, *Operations Research*, Vol. 7, pp 1-10, 1958.

[Stearns, 1988] Stearns S.D., David R.A., *Signal Processing Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[Sukthankar, 1997] Sukthankar R., *Situation Awareness for Tactical Driving*, Ph.D. Thesis, Carnegie Mellon Robotics Institute, 1997.

[Sutton 1998] Sutton R. S., Barto A. G., *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.

[Teleatlas, 1989] TeleAtlas, Vehicle Navigation System and Method, U.S. patent 4,796,191, 1989.

[Tong, 1980] Tong H., Lim K.S., Threshold autoregression, limit cycles and cyclical data, *Journal of the Royal Statistical Society, Series B*, Vol. 42, pp. 245-292, 1980.

[Trimble, 1997] Trimble Navigation Limited, *Lassen-SK8, System Designer Reference Manual*, 1997.

[Turing, 1950] Turing A.M., Computing Machinery And Intelligence, *Journal of the Mind Association*, Vol. 59, No. 236, pp. 433-460, 1950.

[Volkswagen, 2001] Volkswagen web page, available at:

http://www.vwtexas.com/vw_service/Automatic_Operation.asp, 2001.

[Waibel, 1989] Waibel, A., Hanazawa, T. Hinton, G. Shikano, K. and Lang, K., Phoneme recognition using time-delay neural networks, *IEEE Acoustics Speech and Signal Processing*, Vol. 37, No. 3, pp. 328-339. 1989

[Wallace, 1985] Wallace R., Steintz A., Thrope C., Moravec H., Whittaker W., Kanade T., First Results in Robot Road-Following, in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Menlo park, CA, pp 1089-1095, 1985.

[Wan, 1993] Wan E., *Finite Impulse Response Neural Networks with Applications in Time Series Prediction*. Ph.D. dissertation. Stanford University, 1993.

[Werbos, 1974] Werbos P., *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, PhD thesis, Harvard University, 1974.

[Widrow, 1960] Widrow B., Hoff M.E., Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, pp 96-104, 1960.

[Widrow, 1962] Widrow, B., Generalisation and information storage in networks of adaline "neurons", in Yovitz M.C., Jacobi G.T., Goldstein G.D., (editors), *Self-Organising Systems*, Spartan Books, Washington, DC, pp. 435-461. 1962.

[Yang, 1995] Yang L., Widjaja B.K., Prasad R., Application of Hidden Markov Models for Signature Verification, *Pattern Recognition*, Vol. 28, No. 2, pp. 161-169, 1995.

[Yu, 1995] Yu G., Sethi I.K., Road-Following with Continuous Learning, In *Proceedings of the IEEE Intelligent Vehicles*, pp 412-417, 1995.

[Yule, 1927] Yule G.,. On a method of investigating periodicity in disturbed series with special reference to Wolfer's sunspot numbers. *Philosophical Transactions of the Royal Society of London*, No. 226, pp. 267-298, 1927.

[Zhang, 1998] Zhang G., Patuwo E. B., Hu M.Y., Forecasting with Artificial Neural Networks: The State of the Art, *International Journal of Forecasting*, Vol. 14, No. 1, pp. 35-62, 1998.

[Zhao, 1997] Zhao Y., *Vehicle Location and Navigation Systems*, Artech House, Boston, MA,1997.

Appendix A. Publications

In the course of this research we produced the following publications:

1. Mitrovic D., Learning Driving Patterns to Support Navigation Decision Making - Preliminary Results, *Road Safety Conference*, 1998.
2. Mitrovic D., Experiments in Vehicle Movement Prediction, In *Proceedings of the Road Safety Research, Policing and Education Conference*, pp. 519-526, 1999.
3. Mitrovic D., Experiments in Subsymbolic Driving Pattern Prediction, in *Proceedings of the International Conference on Neural Information Processing*, pp. 673-678, 1999.
4. Mitrovic D., Short term Prediction of Vehicle Movements by Neural Networks**,** in *Proceedings of the International Conference on Knowledge-Based Intelligent Information Systems*, pp. 187-190, 1999.
5. Mitrovic D., Driving event recognition by Hidden Markov Models, in *Proceedings of the International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting services*, pp. 110-113. 1999.
6. Mitrovic D., Reliable Method for Driving Events Recognition, accepted for publication in *IEEE Transactions of Intelligent Transportation Systems.* To be published in 2005.