Neural Network Driving with different Sensor Types in a Virtual Environment

Postgraduate Project
Department of Computer Science
University of Auckland
New Zealand

Benjamin Seidler

supervised by Dr Burkhard Wünsche, University of Auckland,
Assoc-Prof Hans Werner Guesgen, University of Auckland, and
Dipl. Inform. Tim Braun, Technische Universität Kaiserslautern

Semester 2, 2006

Abstract

In this project the use of artificial neural networks for autonomous driving tasks is investigated, especially for obstacle avoidance and road following. We have analysed neural networks with input from various sensor types like a single camera, stereo vision, depth information and linear cameras. During the investigation the resulting driving behaviour of the autonomous mobile agent is tested in a virtual environment.
We found that artificial neural networks with input from single sensors result in the best driving behaviour when using depth information. We discovered that combining different sensor inputs with an artificial neural network can generate a better fitting steering output for autonomous mobile agent than with the information of only one sensor. Furthermore, we present interesting results about the influence of varied network topology on the agent's driving behaviour.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Autonomous driving is still a difficult problem. Its avails are for example intelligent transport systems and lateral motion control of vehicles. Therefore, obstacle avoidance and road following are needed.
The control of agents in a complex environment is not easy. Several way finding techniques like retraction and cell dividing exist but they all require prior knowledge of the agent's environment. In map-less navigation there is no explicit representation of the world, like a robot would not have in the real world in a new environment. In this case the agent must rely only on its visual and other sensors. Thus no traditional navigation and path finding techniques can be used. Instead objects have to be recognised and tracking and learning algorithms to identify, extract and observe relevant elements in the environment are necessary as well as geometric reasoning. All these mechanisms have to be realised in real-time when an agent is controlled with this information. Such techniques work well under certain conditions but have many problems with others.
Real world images are full of noise and variability. Real-time, reactive control algorithms that can cope with a high level of sensor data variability and noise are required. Obstacle avoidance in an arbitrary unknown environment or road following on noisy data is a highly nonlinear function between the input (an image) and the output (the steering direction).
Such nonlinear functions are known to be a strength of neural networks.

An artificial neural network is an interconnected group of artificial neurons (units) [7]. It consists of an input layer, zero, one or more hidden layers and an output layer. The units are connected with the next layer and each connection has a weight.
To use an artificial neural network the input is projected to the input units, the activation is propagated through the network and the output units represent the system's answer. This propagation is influenced by the connection weights between the units.
These weights are trained with the back-propagation learning algorithm [40, 45] on training examples. The error of the network's output regarding the desired output is propagated backwards through the network whereas the weights are accordingly slightly modified. Thus the network's error is minimized.
Artificial neural networks have promising performance and flexibility and are capable of handling noise and variability. They are used successfully for handwritten character recognition [16, 29], speech recognition [44] and face recognition [6].

1.2 Motivation

Artificial neural networks can also successfully be used for autonomous driving which was proven by the success of ALVINN, an Autonomous Land Vehicle in a Neural Network [30]. This research showed the flexibility of connectionist learning techniques in autonomous robot navigation. An artificial neural network controlled a test vehicle and learned to follow a road by observing a human driver following public roads for a few minutes [32].
The system could autonomously follow the road under a variety of circumstances like different road types while driving with speeds of over 85 km/h with great success.
These results show that even simple artificial neural networks such as ALVINN can learn quite complex tasks.

1.3 Objectives

The purpose of this project is to investigate the use of artificial neural networks for the processing of inputs from various sensor types like a single camera, stereo vision, depth information (like those from a laser range finder) and linear cameras (cameras with only one line of view) for autonomous driving tasks such as obstacle avoidance and road following. Thereby, it will be analysed which sensor type is most useful. Moreover, different sensor inputs are combined with an artificial neural network that generates the steering output for an autonomous mobile agent in a virtual environment from this information to see if this leads to a better driving behaviour. Furthermore, the network architecture will be varied to investigate what is necessary and what is possible for neural network driving in order to find an optimal setup. For these purposes a virtual environment is sufficient.
Thus, the agent should learn to avoid obstacles and to follow the road as accurately as possible.

1.4 Overview

At the beginning, in section 2, the literature review is given. Previous work related to artificial neural networks, obstacle avoidance, road following and autonomous driving in virtual environments is presented. In section 3 the system's design including the environment for obstacle avoidance and road following, the training of the neural network consisting of gathering training data and training the neural network, and at last the simulation is described. Afterwards, the implementation of these matters is presented in section 4. Finally, the results of the investigations are explained in section 5 for both obstacle avoidance and road following. The work is summarised in a conclusion ,section 6, and possible future work is mentioned in section 7.

Chapter 2

Related Work

A mobile robot in the real world has often no explicit representation of the world, especially in a new environment. The lack of such a map makes traditional navigation and path finding techniques impractical. Therefore, the mobile robot has to rely on its visual and other sensors. Real world images include a lot of noise and variability which makes it difficult to implement rule-based strategies since they have to be robust even on noisy data and changing situations. It is nearly impossible to consider all possible eventualities. Obstacle avoidance in an arbitrary unknown environment on noisy data is a highly nonlinear function between the input (e.g. an image) and the output (steering direction).

2.1 Artificial Neural Networks

The learning of such functions is the strength of neural networks [7]. Artificial neural networks[1] [46] are one of the most effective learning methods for some problems like analysing complex real-world sensor data currently known. Neural networks consist of an interconnected group of artificial neurons/units, where each unit takes a number of real-valued inputs and produces a single output. This is based on real neural networks in brains. A neural network is an adaptive system that changes its bias (the weights of the connections) while learning. Feed-forward networks consist of an input layer, zero, one or more hidden layer(s) and an output layer. The units of each layer are connected to units of the following layer, whereas each connection has a weight.

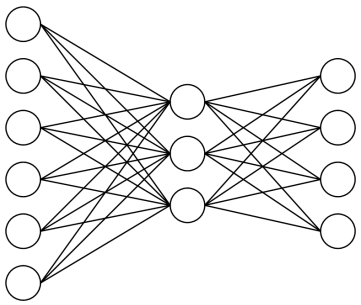

Figure 2.1: A neural network

The neural network processes a given input (values of the input units) and provides a certain output (values of the output units). Each unit calculates the linear combination of its weighted input values and outputs a value regarding a function (mostly if a threshold is obtained). This is propagated through the whole network and so the output of the neural network can be provided. So the output depends on the topology of the network and the weights of the connections which can be changed while learning.

This is usually done by the back-propagation learning algorithm [46] which learns the weights for a neural network given a fixed set of units and connections [45]. It uses gradient descent to minimize the error between the network output and the correct/desired output [40].
Back-propagation is a supervised learning technique. The network is presented an input (training sample), the activation is propagated through the network and the network's response is compared with the correct response; if they do not match, the weights of the units of the output layer causing the error are modified slightly. Then the local error of these units is used to calculate the error

[1] In the following "neural networks" means "artificial neural networks"

of the units at the previous level that are connected with them, and their weights are modified. This process is repeated until the input layer is reached.
Thus for every training sample the weights of the connections are modified to minimize the error of the neural network.

Neural networks are popular in visual perception systems and pattern recognition applications. For example, they are used in handwritten character recognition [16, 29] like the automatic reading of hand written ZIP codes [24, 7]. They are also used in speech recognition [44] and face recognition [6]. Other applications are in control theory [3], data-mining (information filtering [48]), bioengineering (quantitative structure-relation activity prediction for organic molecules [14]) and environment control (ozone pollution and urban hydrology) [7].

In summary neural networks are machine learning tools that are able to learn very general nonlinear functions. Using supervised learning they are high-quality classifiers with a good performance [7]. We therefore infer that neural networks are likely to be useful for obstacle avoidance in unknown environment with noisy data.

2.2 Obstacle Avoidance

Williams [47] showed that a neural network controlled vehicle can find a goal in an obstacle filled environment and is able to avoid these obstacles. The network was trained by manually moving the vehicle through the environment. It was shown that a minimum neural network of 129 input units and 3 output units (forward, right, left) with no hidden units and 400 training pairs were enough to learn to avoid obstacles and even to seek the goal. Nevertheless, the steering of the training data and the control output were simple and not very diverse. Furthermore, the learning took very long.

Aitkenhead and McDonald [1] used a virtual environment to show that a neural network based "animat" (virtual organism) is capable of avoiding obstacles. The animat navigated in the environment relying on its visual sensor system. Its neural network learned to avoid obstacles. The authors tried various network topologies and examined how well the animat learned. By varying the size of the visual input layer and the hidden layer they showed that increasing the layer size improves the performance.
They also investigated the dependency of the system's learning performance upon different training paradigms and possible variations in network topology as well as the learning ability under different environmental conditions. Therefore, a simple world with obstacles and boundaries was created. A vision view of 32 times 24 pixels was the input and the allowed movements were moving straight ahead and turning left or right by a set angle. During training the correct response was determined by a simple rule set considering the distances to an obstacle straight ahead and on the side. The authors tried different training methods like supervised, partially supervised and unsupervised training and different network architectures (changed layer number and size) to estimate the effect on the training time.
They found that increasing input layer size, hidden layer size and number of layers improves learning ability up to a certain point beyond which no improvements are achieved. Also it was shown that neural networks are able to learn obstacle avoidance.

Ruichek [39] showed that neural networks are able to learn obstacle detection with linear stereo vision in real-time. The author used a multilevel neural method for matching edges which were extracted from stereo linear images [38].

There is also a variety of alternative methods used for obstacle avoidance in the literature. Ye et al. [50] used reinforcement learning [23] which has the advantage of not needing supervision during the learning process. A reward is assigned to each action and the system tries to maximize this reward. In this approach in a first step the system learned on training examples with supervised learning. Afterwards fine-tuning was achieved by reinforcement learning. So the system learned more effective collision-free navigation.

A neuro fuzzy controller to control a mobile robot was used by Er and Deng [8]. A generalized dynamic fuzzy neural network (GDFNN) learning algorithm was able to adapt the structure of the controller by itself resulting in very fast learning speeds. The fuzzy rules were automatically generated on-line.

2.3 Road Following

There are also a lot of investigations concerning road following with neural networks. Jeong et al. [17] controlled an autonomous vehicle with a neural network. It was capable of road following with a vision camera. The vanishing point and the vanishing line of a road were used to follow the road marking. This was implemented in computer simulation. First a vehicle dynamical model was set up. Then the transformation of the coordinate systems (world, vehicle, camera and image coordinate system) were calculated and at last the control algorithm was implemented. A neural network (2 input, 50 hidden and one output unit) learned from a human driver and was trained with the back-propagation algorithm. Afterwards it was able to follow straight and curved roads quite well.

The most popular autonomous road follower using a neural network was ALVINN (an Autonomous Land Vehicle in a Neural Network) [30].
Pomerleau used this neural network [32] to control the steering of a modified van (NAVLAB, the Carnegie Mellon's autonomous navigation test vehicle) while driving on public roads to stay on them [33]. A camera in the vehicle produced real-world images which were quite noisy. To implement a rule-based algorithm for road following from this data would be supremely difficult. The result was that even a very small neural network learned such a complex task very quickly.
The author used a single hidden layer feed-forward neural network and the back-propagation learning algorithm. ALVINN got images from a video camera pointed straight ahead or images from a scanning laser range-finder as input. The network consisted of 3 layers (including the input layer). The output was the direction the vehicle should steer to in order to follow the road.
The input layer consisted of a 30×32 unit "retina" which was a low-resolution version of a pre-processed video camera image from a road scene or a laser range finder image. These 960 input units were fully connected to 4 hidden layer units which in turn were fully connected to the output layer. This consisted of 30 units which represented the turn curvature. It was a linear representation of the currently appropriate steering direction in order to keep the vehicle on the road and prevent it from colliding with obstacles. The centre-most output unit represented straight ahead, the more left or right units in turn stood for successively sharper left and right turns.
For driving the image from the sensor was reduced to 30×32 pixels and the intensity of each pixel was projected onto the input layer. The resulting activation of the output layer controlled the steering direction.
The training was implemented by a back-propagation learning algorithm [40, 45]. The video camera image is presented the network as an input. The activation is propagated through each layer of the network and the network's response is compared to the correct driving direction, determined by the driver's steering direction. If these two do not match, the weights of the neural network will be modified slightly. The desired output vector was not only the correct unit activated while all the others were deactivated but a "hill" of activation centred around the unit representing the correct turn curvature to make learning more efficient.
ALVINN was successfully tested on the Carnegie Mellon autonomous navigation test vehicle. It was shown that a neural network can effectively follow real roads in autonomous control under certain field conditions with only a few minutes learning. That worked well under a variety of circumstances such as several different road types. ALVINN was able to control the vehicle with speeds of over 85 km/h. They showed that such a simple neural network can learn a complex task like road following under real conditions. It learned the features which are required for driving on particular road types during training. The system was computationally simple and worked well

in a variety of situations.

Also other learning methods to train ALVINN have been tried. Batavia et al. [4] used Quickprop [10], Cascade Correlation [11], and Cascade 2 while learning road following with a neural network and compared them to back-propagation. Quickprop was even faster than back-propagation whereas the other two of them did not do as well. There was even a hidden layer unit analysis performed to determine what the network learned.

Since a short training of ALVINN did not cover all driving situations and especially rare situations had a lack of coverage in the training set such situations resulted in erratic driving. Pomerleau [31] modelled the appearance of infrequent scenarios by augmenting the training set with important situations that were missing during training. Since the network was trained over a relatively short stretch of road it did not expose to all possible situations during training. Situations like passing cars or guardrails (while driving over a bridge) were rare and limited in duration so there were only very few training samples including them. Therefore, the network was not able to learn how to behave in such situations and was likely to make mistakes. To work against this insufficient diversity in the training examples they tried to teach the neural network to generalize to situations which were not explicit in training set.
First they added Gaussian noise to the training examples but that did not work very well. Training with structured noise was more successful. Physical objects like cars and guardrails are 2d-regions of nearly uniform intensity in the video image. So they added random features to the input images of the training examples, mostly in the periphery. Features were added on random images with a random position, colour and intensity, and then removed randomly.
The neural network trained with structured noise steered more accurately. It learned to rely less on features in periphery and used primarily the lines for following the road. This led to a significant increase in steering accuracy and improved the driving performance, especially when cars passed or while driving over a bridge.

Pomerleau [34] also tried other ways of adding diversity to training data. The performance was improved by transforming the input images and with buffering of the training examples.
The transformation of the sensor images has been done to learn how to correct from swerving without having the driver to swerve himself and to switch on and off the learning process since this would not only be impractical but also dangerous to do in real traffic. So they transformed (shifted and rotated) the input images with an algorithm to obtain images as if the vehicle was facing more sidewards and if it was shifted more to the side of the road. So they gained 15 images out of one. Then the missing pixels were extrapolated and the correct steering direction for the transformed images was calculated. All these samples were added to the training data. Thus ALVINN was able to learn how to behave when not exactly in the centre of the road or not driving straight ahead and how to correct from mistakes without really drive the vehicle that way.
Their next way to add diversity to the training examples they used was buffering. Older training examples were reused for learning again with a certain probability regarding their steering direction. The purpose was to to keep the mean steering direction of the buffered data set straight ahead so that ALVINN did not learn to prefer one of the directions.
With these techniques they improved the performance of ALVINN and showed that adding diversity of training data is quite important.

Baluja [2] used an evolutionary method for creating artificial neural networks that should control an autonomous land vehicle to follow a road. The goal was to find networks with good generalization ability with evolutionary algorithms.
The back-propagation learning algorithm is a gradient descent which searches for a local minimum. Evolutionary algorithms in contrast perform a global search and therefore, are less susceptible to keep stuck in a local minimum. The author used evolutionary optimization methods to improve generalization capabilities of the feed-forward neural networks. The network topology was discovered as well as the weight were optimized with the aid of evolutionary algorithms. Since these are computationally very expensive the whole network pool has to be trained before any use of the learned neural network, so no on-line training was possible.

The advantage was in combining evolutionary search and back-propagation: First a population based incremental learning was used to learn both topology and weights for the neural network (within certain limits like number of units and layers) and afterwards the weights of this network were refined by using back-propagation.
The so achieved road following system performed better (on NAVLAB) in unseen situations than those trained only with back-propagation.

Another way to obtain a more accurate neural network is reinforcement learning [23]. Yu and Sethi [51] used a neural network based road follower with vision sensor input that generated the steering control output. The network was not only trained in a supervised mode but also with reinforcement learning. Thus the authors eliminated the need for external supervision and added continuous learning ability (like human drive practice) to the system.
Reinforcement learning is an unsupervised learning method in which the system is given a feedback (reinforcement) systematically. This performance score is either a reward/payoff or a punishment/critic regarding if the action was good or bad. With this method the neural network's connection weights were modified. Thus the reinforcement was maximized to improve its future performance.
With the combination of a neural network and reinforcement learning which reflects the human driving practice the system was capable to learn continuously how to generate the correct steering output.

Another reinforcement learning road follower was presented by Oh et al. [27]. The new dynamic control architecture was capable of high-speed road following of high-curvature roads. Reinforcement learning was used to handle image features which were first extracted during an image processing step.

As seen above a neural network can drive a vehicle reliably and safely on many different road types reaching from paved paths to multi-lane interstate highways. But what was missing in ALVINN was the transparent navigation between different road types, the simultaneous use of different sensors and the generalization to former unseen road types. ALVINN is good for one type of road, but only for roads it has already learned. Pre-trained networks existed for several road types but the system had problems with new ones.
MANIAC (Multiple ALVINN Networks In Autonomous Control) [19] was a modular neural architecture which tackled these disadvantages. It allowed transparent navigation between roads of different types using pre-trained ALVINN networks. With a neural network combining pre-trained networks the system was able to drive robustly on many different types of roads. The authors hoped that MANIAC would learn to combine the pre-trained networks to gain new information instead of simply selecting one of them.
The output of the hidden units of multiple ALVINN networks, each pre-trained for a particular road type, were connected to the input of the MANIAC network. This one was trained using back-propagation with images from different road types which were permuted.
The trained system was in a simulator with different road types, those ones the partial networks were trained on and also new ones. The combined neural network (MANIAC) worked better on a new road than both of the used pre-trained networks, and did not work much worse than the particular networks on the road types they had been trained on.
A system like this can also be useful for incorporating information from different sources.

Other features which were added to ALVINN include vision guided lane transition [22], intersection detection [21], traversal [20] and navigation [18].

However there were also other systems. ROBIN (Radial Basis Network) [37] was a template-based visual autonomous road-following system which was based on a radial basis function architecture. A neural network was the inner reactive component of the system. It was trained by a human operator on a specific road type. These pre-trained instantiations were saved to activate them again later. Diverse sensors like a colour camera, a FLIR and a low light camera were used. Thus the system was able to navigate successfully on a diverse set of roads reaching from worn tracks in the grass to multi lane highways.

Another System developed by Pomerleau and Jochem [35] was "Ralph", a vision system designed to help automobile drivers to steer. It got a sample image, assessed the road curvature and determined the lateral offset of the vehicle relative to the lane centre. With this information it was able to determine the correct steering direction. The system performed well, for example during a coast-to-coast, 2,850 mile drive.

2.4 Autonomous Driving in Virtual Environments

For the purpose of testing and evaluating algorithms for autonomous driving or even for pre-training neural networks several simulators were developed.
One of them is SHIVA, the Simulated Highway for Intelligent Vehicle Algorithms [43], which focused tactical driving on highways [41] mainly for intelligent transportation systems [42].
Since testing new algorithms in human traffic is risky and potentially dangerous simulation of them is essential. SHIVA provided a realistic sensor model and driver models as well as an efficient integration for real robotics since the interfaces were quite similar to those in the NAVLAB test vehicle. The roads consisted of an arbitrary number of lanes. Different types of vehicles, controllers, sensors and driver models were included. The user interface provided different displays and views like bird's eye, road side or driver's eye view.
With the help of such a simulator it is possible to test the algorithms designed for real traffic safely in a virtual environment without endangering oneself or others. Other road simulators are Smart-Path [9] and TRAF-NETSIM [49].

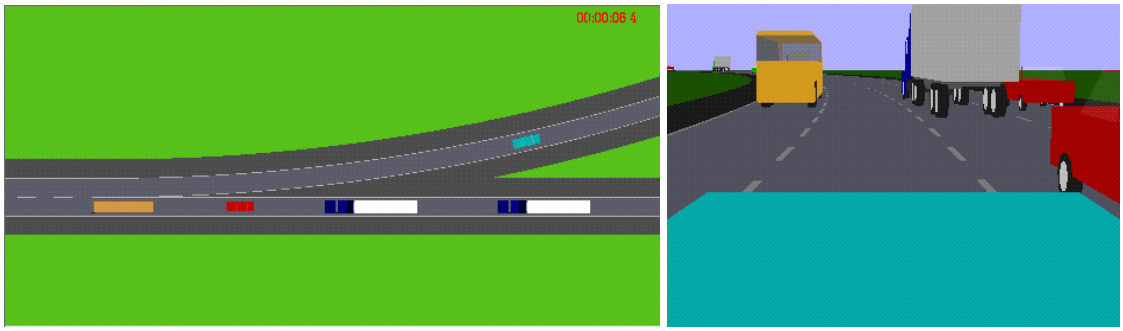

Figure 2.2: SHIVA (Simulated Highway for Intelligent Vehicle Algorithms) road simulator

Autonomous driving in a virtual environment was done by Flower et al. [12, 13]. A mobile agent was controlled by a neural network getting only the information the agent could see from its point of view. The autonomous agent was put in a interactive virtual environment and trained to follow the road or to find a path to a given goal. Therefore, a virtual city with intersections, traffic lights, bridges and buildings was created. The autonomous agent learned to follow the road and to find a path in a virtual environment if certain conditions were fulfilled. This was investigated for a possible usage in computer games.

Altogether, many approaches for obstacle avoidance and road following with neural networks exist. However, only in some of them variations to the neural network topology were mentioned. In none of the cases different sensor types were compared or was tried to combine different sensor types in order to compare them and achieve better results for autonomous driving tasks like obstacle avoidance and road following.

Chapter 3

Design

This section describes the design of the system to investigate autonomous driving with neural networks. The system consists of an virtual environment wherein a human driver can record information on which a neural network can be trained. This controls the agent for the tasks of obstacle avoidance or road following.

The work of this project is based on a previous project of Daniel Flower [12]. He created a virtual environment for autonomous agents with simulated vision controlled by a neural network [13]. A mobile agent had to rely only on what is saw with a mounted camera facing straight forward. It was put in an interactive environment with the idea of a possible usage in computer games. Different investigations like obstacle avoidance, road following and path finding have been done. It was analysed how well the autonomous agent navigates its way through a virtual city with intersections, traffic lights, bridges and buildings by only using what it sees from its point of view. This computer generated image with quite simple complexity was used as input of a neural network which produced the direction to steer as output. The system learned from training examples saved while a human driver steered the virtual agent. After training the neural network could be activated to control the agent's steering direction autonomously. It was shown that autonomous agents can learn how to avoid obstacles, follow the road and find a path in a virtual environment under certain conditions.

The objective of this project is to use various kinds of sensors like a single camera, stereo vision, depth information (as provided from a laser range finder) and linear cameras as input for a neural network. An autonomous agent is trained to drive autonomously in a virtual environment.
Additionally, distinct sensors will be consolidated with a neural network to generate the steering output. Thus the agent should learn obstacle avoidance and road following. Furthermore different neural network architectures and other variations on the training setup will be investigated.

This section provides an overview of the system's design. The environment for obstacle avoidance and road following is described in section 3.1. In section 3.2 is explained how the training data is gathered and in section 3.3 how the neural network is trained. Finally, section 3.4 deals with the simulation of the neural network driving the virtual vehicle autonomous.

3.1 Environment

For the tasks of obstacle avoidance and road following a virtual environment was created (see figure 3.1). In order to investigate different types of sensors and the ability to follow a road in a virtual environment this had not to be very realistic since graphic details like textures are not important for that (a human driver would not need a high degree of detail for the mentioned tasks). Thus, many things could be kept simple, e.g. a flat world without hills and slopes was sufficient. The graphics could be simple as well since a simple environment already contains enough information

for the desired investigations. So more complex graphics including textures and models are not necessary. For a realistic simulation of the real world and a view with very high resolution for the agent a more realistic environment could be necessary. However, analysis of high resolution images is not topic of this project and therefore not considered here.

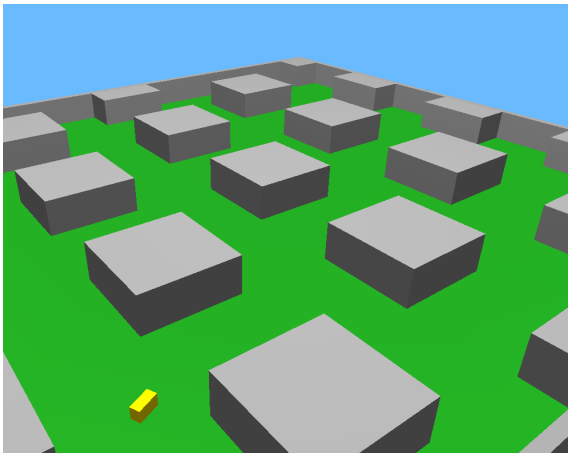

Figure 3.1: Virtual environment

To render the environment from the agent's point of view different sensors can be simulated. Besides the normal view as it would come from a single vision camera, stereo vision cameras, depth information (like from a laser range finder) and linear cameras (cameras which record only one line) can be used. For the agent's view the perspective image from the environment is reduced to a much lower resolution since this information should be enough to avoid obstacles and to follow the road (a human driver could do this with a reduced view as well, see figure 3.2). Furthermore, the image is converted to greyscale (see sections 3.2 and 4.1 for details). This view can be displayed on the screen and while recording data to learn (see section 3.2) these pixel values are written into a file.

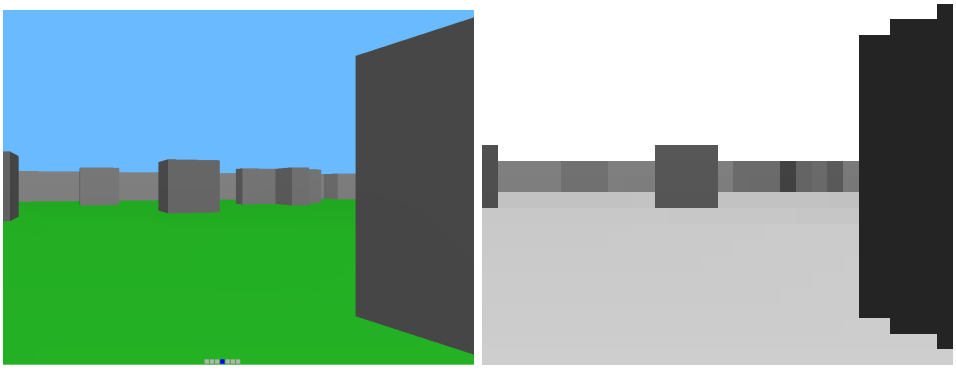

Figure 3.2: Scene from the agent's point of view; original left, reduced right

The user's view can be the one of the agent but also a freely movable camera view or a bird's eye view. The path taken by the agent can be displayed for evaluation purposes. The agent itself only knows what it sees from its point of view and has no map or other prior information about the environment. This provides the same situation as for a human driver or a real robot in unknown environment and leads to realistic driving.

To learn *obstacle avoidance* a simple world with obstacles was created. This is just a bounded area (enclosed by border walls) and simple obstacles with different shape, size, position, and rotation (see figure 3.3). Different maps were used for training and testing the neural network.

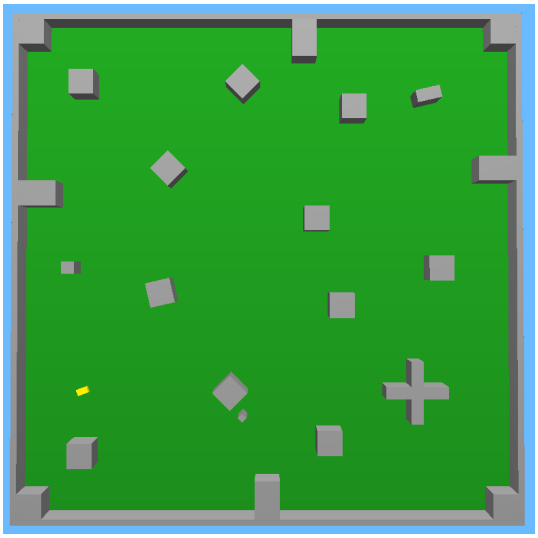

Figure 3.3: An obstacle avoidance map

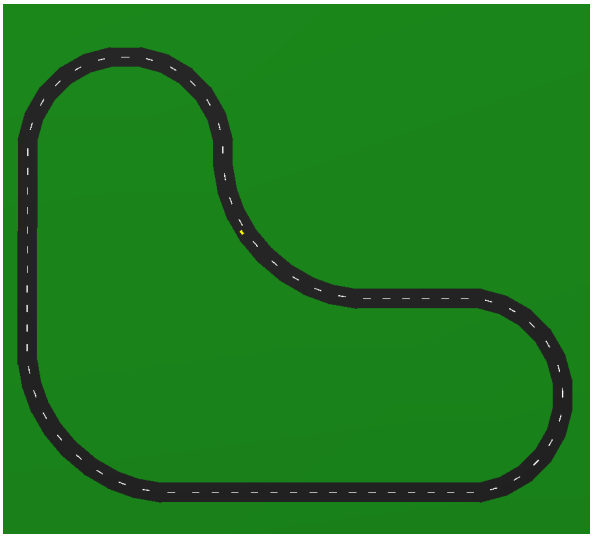

Figure 3.4: A road following map

For *road following* the environment is simple as well. To investigate the ability of following the road and improvements thereupon with image transformation methods, a curved road on flat ground as in figure 3.4 is sufficient. To evaluate the neural networks other road maps or just the same road in the other direction can be used.

3.2 Gathering Training Data

To train the neural network to avoid obstacles or to follow the road training data from a human driver are recorded. The agent should learn to imitate the driver's behaviour without telling it explicitly rules to follow or features to react on. The human driver can control the agent with the keyboard like in a race simulation in order to show the agent how to drive.
For obstacle avoidance the driver just drives around avoiding obstacles. If an obstacle is in the way he turns the vehicle. The driver can react quite early or late to obstacles in front of him and the agent should learn to behave similar to that.
For road following the human driver should follow the road, thus staying in the centre of the road or the lane, respectively, as exact as possible.

Training data is recorded for approximately four minutes for obstacle avoidance and even shorter for road following. The pixel information of the reduced view of the agent and the current steering direction is stored in a file every few frames. For this controllable screenshot rate a fixed time of approximately five examples per second is appropriate since with more examples per second many pictures would look the same and therefore would not contain much new information. Thus, there is more diversity in the training data of the same size with less screenshots per second.
The image quality is reduced since much less pixels than there are on the screen are needed to learn obstacle avoidance or road following and too much pixels would slow down the learning process. The number of pixels in the agent's view is controllable in order to test how different view sizes work.
Furthermore, the image is converted to greyscale. There is not much more information which is necessary for driving contained in the colour of the image than in the greyscale image in general

as showed in [34]. A human driver is able to drive a vehicle and avoid obstacles or follow the road without colour, too. Therefore, colour is not used for autonomous driving in this project.
As colour is not needed for these tasks it is appropriate to convert images the agent gets to greyscale. The same holds for the resolution; a human driver would also be capable of driving a vehicle with a low resolution image of the environment. But can a neural network even drive with less information than a human driver would probably need? This issue will be investigated with sensor images with very low resolution.

The agent can be controlled by a human driver like in a car racing computer game. The steering and the velocity can be affected with the keyboard. Since the angular velocity and thereby the curve radius is influenced the driver can drive long turns and sharp turns where it is needed. These differentiated steering values (different sharpness of turning) are stored in a file together with the sensor information and are used to train the neural network. Additionally, the previous steering value is stored as well since this can be useful, e.g. if driving straight towards a wall or an obstacle it can be possible that the neural network controlled agent alternates between left and right and does not decide for one direction. In that case it can be an advantage to use the last steering direction to help the agent to select one direction and avoid the obstacle.

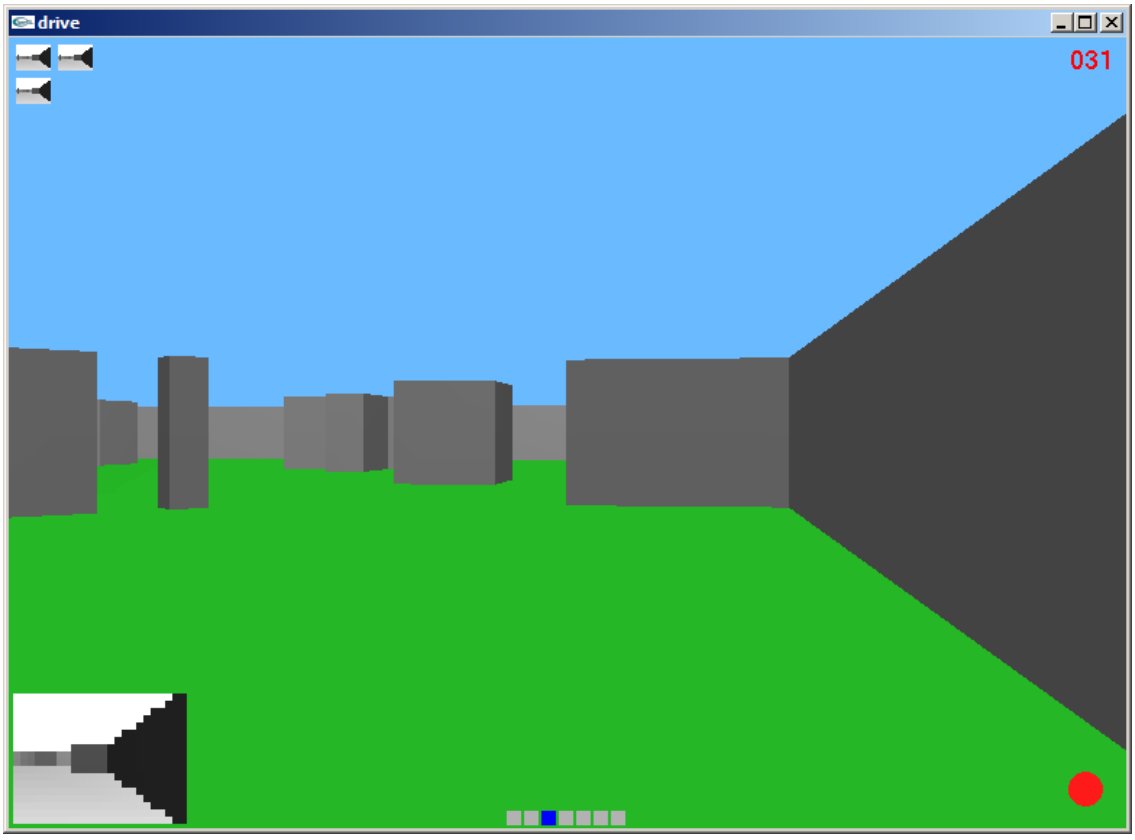

Figure 3.5: A human driver is recording data in the agent's view

Figure 3.5 shows the agent's view while a human driver is recording training data. The red circle in the lower right corner indicates that data is currently recorded, the time counter in the upper right corner shows how long training data has been recorded yet. The blue point in the middle shows the current steering direction. The recorded data of the agent's view is visualised in the lower left corner.

While training the neural network, the human driver has to consider that the neural network will imitate his behaviour. If he avoids obstacles in the nick of time the agent will do this as well and if he turns much more often to one side the agent will also be biased to this direction. The same

holds for road following: when there are much more turns to one side in the street and so in the training data the agent will be more likely to drive to this direction after training.

For *obstacle avoidance* several sensor types are recorded in parallel. These are a single vision camera, two stereo vision cameras, depth information (like from a laser range finder) and linear cameras (cameras with only one line of view). The single vision camera is just the reduced image of the scene from the agent's point of view. For the stereo vision two cameras at the side boundaries of the virtual vehicle are simulated and both of their views are used. The depth information is read out of the depth buffer of the single vision camera's view. These are greyscale values representing the distance of an object. The linear cameras are just particular lines of the single camera's view and the stereo cameras' views, respectively. The pixel information of each sensor is stored in a file with the information from which sensor type they come. They are all written to the same file in order to compare different sensor types with the same training data and to combine different sensor types in one neural network.

For *road following* the vision camera information is used. This is stored in a file together with the current steering information of the human driver. If only the values which were recorded while driving in the centre of the road are used the neural network will never be presented training examples showing how to correct from mistakes. Thus, it cannot learn how to steer back to the road if it made a mistake. To approach this problem it would be possible to turn the learning process off, leave the centre of the road, and record data how to drive back to the road. However, this is very time consuming since it has to be done quite often to get enough training examples. A better possibility was introduced by Pomerleau [34] who transformed the images. Of each training image multiple versions were created namely shifted and rotated images. This is quite difficult in the real world since missing pixels have to be calculated whereas in a virtual environment the virtual camera has only to be shifted and rotated before rendering the view and storing the data. This additional image information is used for training as well. For each image the appropriate steering direction is needed, too. Obviously it is not the same as the steering direction of the normal view. Thus, a modified one has to be computed. This is realised by a function named "pure pursuit" which gets the shift and rotation values and the correct steering direction for the normal view as well as a look-ahead value. The look-ahead is the distance after which the virtual vehicle should be at the same point as the one with the normal view and the real steering value, e.g. in the centre of the road. With these values and the current steering radius a formula delivers the appropriate steering radius for the transformed view (for more details see section 4.2.1). The output representation of this value is stored with the transformed image information in the file.

To train the neural network on this information the appropriate steering direction for each recorded or transformed image is required as well. A representation of the steering value is also stored in the same file.
The number of output units representing the steering direction is predefined but controllable. The leftmost output unit represents a sharp left turn, the median output unit straight ahead and the rightmost output unit a sharp right turn (the units between not so sharp left and right turns) (see [34].
Instead of just storing a one for the current steering direction and a zero for all the others (here all neural network values are between zero and one), a Gaussian curve around the current steering direction where the mean (μ) is the current steering direction and the variance (σ^2) depends on the total number of output units, is used. This makes sense since a turn without the exact sharpness but the same direction is normally also an appropriate steering value and therefore not to be set to zero in the learning data. In this way a smooth distribution of activation around the current steering direction is stored.

All the stored values are real values between zero and one. For the sensor inputs this is the normalized brightness information of each pixel and for the output the normalized steering value with the Gaussian curve of activation. Taking only values between zero and one for both all inputs and the output allows to learn the neural network easier and faster.

All possible input values and the desired output values are stored in a file. To be able to record

more training examples bit by bit and to allow to delete one file when the human driver made a mistake each recording session creates a new file with the current date and time in the filename. This contains all information that is needed to train the neural network.

3.3 Training the Neural Network

What a neural network is and how it works has already been explained in sections 1.1 and 2.1. For further information the reader is referred to the literature, e.g. [7] and [45].

With the recorded training examples a neural network can be trained. The inputs (the sensor information) and the correct output (the correct steering direction) for these inputs are stored in a file as explained above. The neural network is trained with this data and should learn obstacle avoidance or road following. Therefore, no further information about the environment or any rules are needed. The network should learn the features needed to avoid obstacles or to follow the road just from the training data.

The structure of the neural network depends on the sensor types which are used and the number of output units. Since different values are to be compared both the current sensor types and the number of output units can be controlled. In the majority of cases a fully connected neural network is used. The used sensor types and the size of the agent's view determine the number of input units. Which pixel is used as activation for which input unit is arbitrary as long as the same mapping is used for training and simulation. The number of hidden units and layers, if any, can be controlled in order to compare different settings, too.
It is also possible to train two neural networks to get steering directions for a stereo camera, one for each input. These two have to be consolidated when the neural network is used for autonomous driving.

The weights of the neural network are learned with the training examples and the chosen configuration using the back-propagation learning algorithm [45]. Back-propagation uses gradient descent to minimize the error of the neural network [40]. It is a supervised learning technique where the neural network is presented an input (here the reduced image of a sensor), the activation is propagated forward through the network and the network's output is compared to the correct response (the demanded steering direction). If the output and the correct response differ the weights of the output layer causing the error are modified slightly. Then the local error of these units is used to correct the weights of the previous layer and so on. So the errors are back-propagated through the neural network and all the weights of the connections can be modified in order to minimize the global error of the network for this training example. This is done for every training sample to minimize the error for the whole training set. The parameter with which the weights are modified can be controlled but it can also change during the learning process if e.g. "momentum" is used.

For learning obstacle avoidance and road following *iRprop*, a variation of *RPROP* [36] developed by Igel and Hüsken [15] is used. This learning algorithm works similar to back-propagation with momentum but uses so-called "learning epochs" and does not depend on the derivative value but only on its sign. This leads to a higher learning speed and faster convergence times. For more information about these learning methods see section 4.2.5.

Before learning the weights of the neural network these should be initialised. Instead of assigning a zero as weight for each connection, random values result in faster and more accurate learning as showed later. Here, first random values and then the Nguyen-Widrow layer initialisation function [25] were used for weight initialisation. This method starts with random weights which are modified later. See section 4.2.3 for more details.

In Summary, a neural network with the chosen configuration can be trained on different sensor information with given view size. Different network architectures and even two neural networks for stereo can be used as showed later. This allows comparing various different settings of sensors and neural networks in obstacle avoidance and road following. The trained neural network can be saved for using it in the simulation where it can be loaded and run again.

3.4 Simulation

Testing the trained neural network works quite similar to gathering the training data (see section 3.2), except that the virtual vehicle is not controlled by a human driver but by the neural network. The path taken by the agent so far can be drawn for evaluation (see figure 3.6).

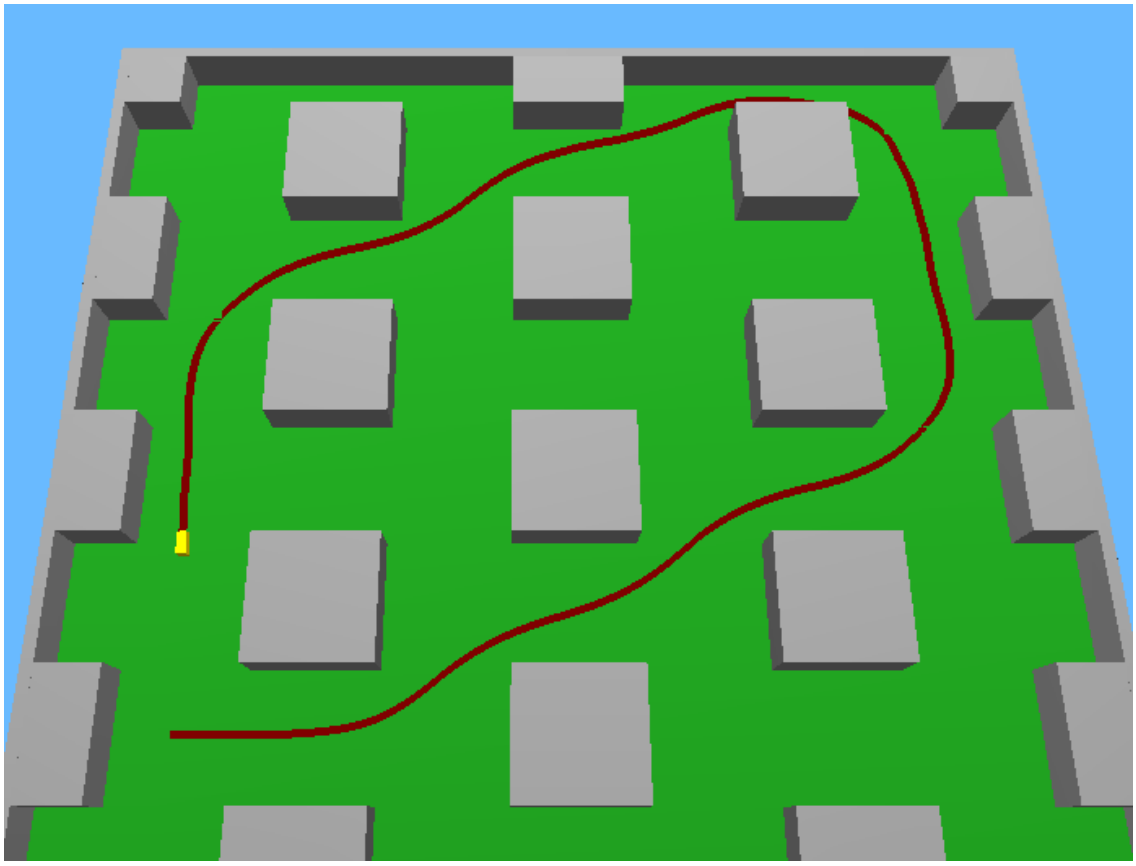

Figure 3.6: Agent drives autonomously and avoids obstacles

The trained and saved neural network is loaded. Every frame rendered by the current scene from the agent's view (reduced and converted) is given to the network as input, the same way as it was trained with. The values depend on the brightness (or depth) of the pixels of the current view. This activation is propagated forward through the network using the trained weights (these are not changed anymore). As result the output units have a certain activation which is used to determine the steering value from.

Each output unit represents a steering direction as explained earlier. To gain one appropriate steering direction from the activation of the output units miscellaneous possibilities exist. One would be to take just the output unit with the maximal activation value as steering direction. This is fast and easy but leads to discrete steering values depending on the number of output units. Other possibilities would be to take the weighted average of all output units or to calculate the best fitting Gaussian curve over the output units and take its mean/centre as steering direction. Similarly, only the best fitting Gaussian curve next to the maximal value could be used as steering direction. All these techniques will be explained in more detail in section 4.3 and evaluated in section 5.1.4.

Different neural networks trained before with diverse inputs, and varying hidden layer size and even two neural networks to process stereo information can be used in the simulation. In the latter

case both networks had to be trained on the same data using only one half of the recorded stereo information. Table 3.1 gives an overview of the different setups.

single sensors	vision camera, depth information, stereo cameras, linear camera(s)
stereo cameras	different ways to combine stereo vision information
combining sensors	vision camera + depth information, depth info + linear camera
view size	different amounts of pixels per sensor image
network architecture	number of hidden units, connection rate, weight initialisation
output representation	maximum, weighted average, best fitting Gaussian curve (near max.)

Table 3.1: Overview of the investigated setups

If the training data was biased to one direction the neural network will also be more likely to drive into this direction since it learned driving to this direction more often than to the other one from this training data. Diversity in the training examples is important and the human driver should balance right and left turns while recording training data.

To improve a biased training set or to add training examples which are not in the training set yet it is possible to record data during testing. Therefore, the neural network is not used during that time and the human driver takes control of the virtual vehicle again. These new training examples (together with the old ones) can be used to train the neural network again.
Another idea to avoid the bias in the training set would be add the mirrored images and the appropriate steering values to the training set. However, this was not used here and is future work.

In general, the neural network should be able to generalise to new situations after the learning process. Hence, testing can take place in unseen environment, e.g. a new obstacle layout or another road for obstacle avoidance and road following, respectively. The neural network should have learned features which are important for obstacle avoidance or road following that it has not been explicitly told before, so that it can be used in new situations as well.

Chapter 4

Implementation

This section gives an overview of the implementation of the system. The realisation of the environment for obstacle avoidance and road following (section 4.1), the neural network and training algorithms (section 4.2) and the simulation (section 4.3) are described.

The system consists of two programs: The `drive` program used for gathering the training data and simulating the neural network and the `learn` program which trains the neural network's weights with the training data. The `drive` program is written in `C++` using OpenGL[1] with GLUT (OpenGL Utility Toolkit). The `learn` program is a `C++` command line tool. For the neural network implementation *FANN*[2] (Fast Artificial Neural Network Library [26]), a free open source neural network library in `C`, was used.
Object-oriented programming was used for this system. The `drive` program consists of a `main` class which reads in the configuration file[3], sets up OpenGL and GLUT information (create window, set up light, camera, key listeners etc.) and creates a `Robot` (a virtual agent) which is processed every frame.

The `Robot` changes its position and rotation (and so its view) depending on if it is controlled by a human driver or by a neural network. In the former case the velocity and the angular velocity is controlled by the user with the keyboard. In the latter case a neural network is loaded and used for determining the steering direction. The inputs for the neural network are saved in an array by a method in the `AgentView` class. The `Robot` interprets the outputs of the neural network an determines the next steering direction. Afterwards the movement is checked for collisions with the environment.

The input for the neural network is the current view of the agent which comes from the `AgentView` class. When a neural network controls the agent, `AgentView` updates an array with the sensor input for the neural network every frame. Therefore, the scene from the agent's point of view is rendered in sub-windows depending on the current used sensors. The pixel information is stored in an array and processed with the neural network to get the next steering direction.
When in contrast a human driver is controlling the agent and recording training data the sensor information is obtained in the same way but written to a file instead of an array. This file can be used for training the neural network later.

Many settings can be defined in the config file. It contains general information as the window resolution, if the agent is controlled by the keyboard, the size of the agent's view, the used sensors, which map should be loaded and folder and filenames but as well settings referring to training the neural network like the number of hidden units, the connection rate and the number of epochs to learn. Furthermore, the config file includes information about the simulation of the

[1] Open Graphics Library, see `www.opengl.org/`
[2] Available from `http://leenissen.dk/fann/` as at August 2006
[3] The configuration file will in the following be called "config file".

neural network like the file name of the neural network and whether two networks should be used for stereo cameras. For road following it can be controlled if white stripes in the middle of the road should be used, if the images should be transformed, and the look-ahead distance. This high degree of ability to configure the programs is very helpful for many tests with different settings.

The scene itself depends on the map which is a file in a certain format read in by the `MapReader` that generates objects and their geometry.

4.1 Environment

As explained in section 3.1 the virtual environment can be kept simple and must not be very realistic for the desired investigations. A simple world already contains enough information for obstacle avoidance with different sensors and road following since the resolution of the agent's view is too low to obtain detailed information from its environment. However, a realistic view of this simple environment is important. Therefore, perspective is used for the rendered scene from the agent's point of view. The illumination is both diffuse and ambient, the background colour is light blue for the driver's view and white for the agent's view since this makes distinguishing between background and obstacles easier. All these settings are applied to the scene using OpenGL.

Which objects ought to be in the scene is controlled with the map file loaded from the `MapReader`. Such a file is a list of object types followed by their position, rotation, colour and size. For each entry the object and its geometry is created and then they are added to the scene.
Every object in the world is a `WorldObject`. Special cases of that are `StaticObjects` like obstacles, walls, and `Roads` and the `Robot` which can move and whose `process` function is called in every frame.

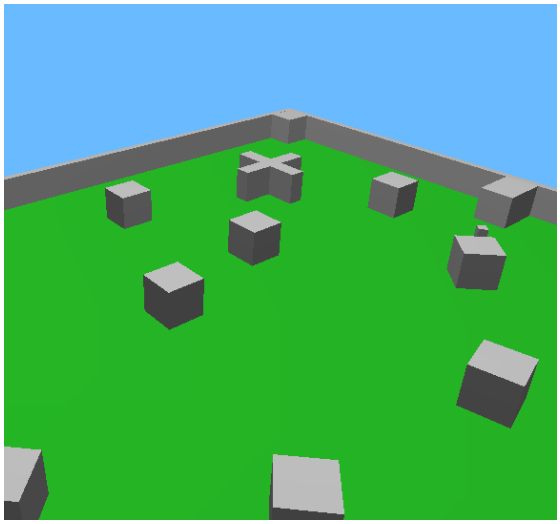

Figure 4.1: Obstacle avoidance ...

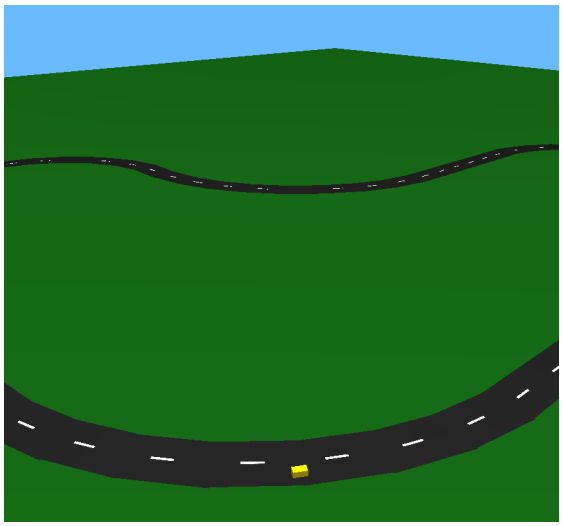

Figure 4.2: ... and road following environment

For *obstacle avoidance* a map consists of a bounded area enclosed by walls with objects of varying shape, size, position, and rotation (see figure 4.1). Different maps are used for gathering training data and testing the neural network.
For *road following* a curved road forms a cycle to make it easier to drive for a longer time (see figure 4.2). This has the advantage that the camera does not have to move when the agent drives for a longer time. The road can be used for training in one direction and for testing in the other. Different road maps are possible as well.

The view of the environment can be varied. Besides the view from the agent's point of view a free camera view and a bird's eye view are possible. These can be controlled with the keyboard.

The scene from the agent's point of view is rendered in sub-windows on the screen. These images are reduced in size and converted to greyscale (see figure 4.3).

While the human driver drives the vehicle through the environment and records data the reduced image information from the agent's point of view for each sensor is stored in a file (see section 4.2). Thereby, the agent only gets the information it sees and has no map or other representation of the environment which would allow an optimal driving behaviour or at least a kind of planning. This leads to a realistic steering and is the same situation as if a real robot was in unseen environment.
The stored information can be used for training the neural network.

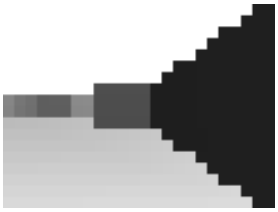

Figure 4.3: Reduced and converted view

4.2 Neural Network Training

In the following the training of the neural network is explained in more detail. First it is explained how the training examples are recorded (section 4.2.1) and then the learning program is described (section 4.2.2). Afterwards, the weight initialisation of the neural network (section 4.2.3) and the standard back-propagation algorithm (section 4.2.4) are presented. Finally, the *RPROP* learning algorithm and its variation *iRprop* (section 4.2.5) which is used here are explained.

4.2.1 Gathering Training Data

To train the neural network training examples have to be gathered first. Therefore, a human driver controls the virtual vehicle to avoid obstacles or to follow the road. The velocity and the angular velocity are controlled with the keyboard which allows longer and sharper turns, respectively. When recording training data the sensor values and the steering value are stored in a file every few frames (in most cases every 30 frames a screenshot is taken, that is approximately between one and 5 per second).

The input information for the neural network are the pixel values of the reduced sensor image that has been converted to greyscale. Therefore, the scene is rendered from the agent's point of view in sub-windows depending on the currently used sensors (for obstacle avoidance). There exists a sub-window for the single camera and two translated camera views for stereo vision, one on the left and one on the right side of the virtual vehicle. From these three rendered views the image information for single and stereo cameras and also the depth information and the values for the single cameras can be obtained. For the depth information the depth buffer of the centred sub-window is used and the linear camera information is just one line of the normal camera view. This sensor information is written to a file together with the current steering value of the agent controlled by a human driver. Therefore, the value of a Gaussian curve around the current steering direction is used for every output unit . The mean (μ) of the Gaussian curve is the current steering value and the variance (σ^2) depends on the total number of output units. Without the factor at the beginning of the formula all values are real and between zero and one which is useful for training the neural network on this data. The formula used here is $x[i] = \exp\left(-\frac{1}{2}(\frac{i-\mu}{\sigma})^2\right)$, whereas $x[i]$ is the stored activation value for output unit i, μ is the given steering direction and σ^2 is one tenth of the total number of output units.

Figure 4.4: The shifted and rotated views

For *road following* shifted and rotated image information is stored besides the normal view. To get more training data, to show the neural network how to correct mistakes and how to find the way back to the centre of the road (since it would not have been presented training examples showing this otherwise), the normal vision camera image is transformed to get 17 images out of one (see figure 4.4). Therefore, the virtual camera is shifted and rotated each to the left and the right by two values. All this can be done with `OpenGL` wherefore no missing pixels have to be approximated as it would be necessary when using a real camera since the images and not the camera have to be transformed and therefore not all needed data exists. The different views are used as training examples in turn, thus not every frame the values of all transformed images are written to the file but only one of them. Each view is rendered only when its values are recorded.
For these image values the appropriate steering direction is still missing to be complete training examples. This direction is computed[4] with the shift s and the rotation θ of the image together with the current steering radius r_h of the human driver and a look-ahead distance l. This latter value describes the distance to the point where the vehicle from the original point driving with the real steering value and a vehicle driving from the transformed position with the steering radius which will be computed should be at the same place again. Therefore, the desired steering radius needs to be chosen.
With these values the quantity d_h (see figure 4.5) can be calculated as $d_h = r_h - \sqrt{r_h^2 - l^2}$. With this for the displacement d the equation $d = \cos\theta \cdot (d_h + s + l\tan\theta)$ holds. Now it is possible to calculate the desired steering radius r as $r = \frac{l^2+d^2}{2d}$. This has to be transformed back to a steering value. The representation of the computed steering direction as an output value is stored in the file together with the pixel information of the appropriate transformed image.

The frequency of writing examples (input and output values) to the file is controllable in the config file. In this project between one and five examples per second are written. The size of the data depends on the current size of the agent's view which is controllable as well. One training example containing all sensor information and the output representation looks like this (here with an agent's view size of three times two pixels):

```
NORMAL 1 1 1 0.792157 0.792157 0.101961
STEREO 1 1 1 0.792157 0.792157 0.101961 1 1 1 0.792157 0.792157 0.101961
DEPTH  1 0.886621 0.860191 0.900477 0.886621 0.860191
LINE   0.792157 0.792157 0.101961
LINE2  0.792157 0.792157 0.101961 0.792157 0.792157 0.101961
OUTPUT 0.226815 0.953571 0.542557 0.041778 0.000435373
```

Each line contains a sensor name (or `OUTPUT`) with the appropriate pixel (or depth) values. `NORMAL` is followed by the pixel values of the single vision camera, `STEREO` by those of both stereo cameras, `DEPTH` by the depth information and `LINE` and `LINE2` by the linear camera information for a single and two stereo cameras, respectively. `OUTPUT` is followed by the output representation as described above. All values are real values between zero and one.

[4] This method was explained by Pomerleau in [34]

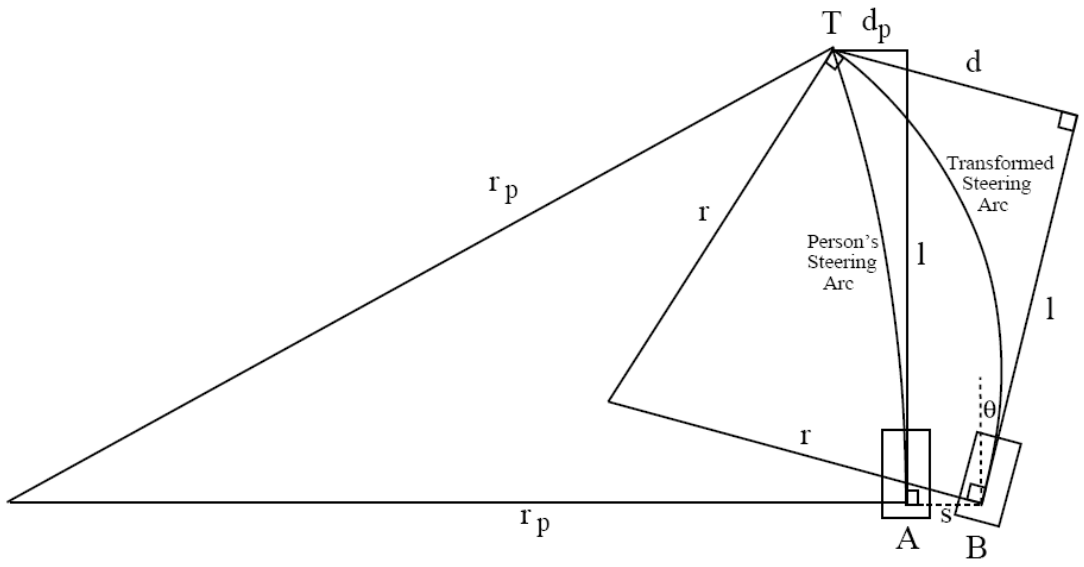

Figure 4.5: How the steering radius for the transformed images is calculated (image from [34])

4.2.2 The Learning Program

With this recorded information a neural network can be trained. This takes place in the `learn` program, a command line tool which uses the recorded data and the parameters from the config file to train a neural network. The number of epochs to train (each epoch the training data is presented once to the neural network), the desired error (specifying when the training should stop) and the architecture of the neural network can be controlled.

For the neural network implementation *FANN*[5] (Fast Artificial Neural Network Library [26]), a free open source neural network library in `C`, was used. It supports multilayer neural networks and back-propagation training and is quite fast. Training and saving the neural network and also loading and running the network in the simulation is done with *FANN*.
To use *FANN* the training examples first have to be converted to an appropriate file format containing only the information from that sensor types that should be learned now. If the neural network should be trained with more than one recorded file all the training examples of the files in a folder are combined. However, before training the neural network on this preprocessed data the weights of the connections in the neural network should be initialised.

4.2.3 Weight Initialisation

Weight initialisation is important since it improves the learning process. If this is done reasonable learning will be a lot faster. Thereto, random values can be chosen. This works in most cases better than an uninitialized network, e.g. all weights set to zero. The learning is usually faster and the behaviour improved.
For a fully connected two layer neural network exists a method that leads in general to even better results. The Nguyen-Widrow layer initialisation function [25] sets weights for each layer so that the active regions of the layer's units are distributed approximately evenly over all possible input values. The main idea of this algorithm is to initialise the weights with small random values. These are then modified in order to divide the region of interest into small intervals. To make training faster it is reasonable to initialise the weights of the first layer so that each node is assigned its

[5] Available from `http://leenissen.dk/fann/` as at August 2006

own interval at the beginning. During the network's training each hidden unit can still adjust its interval size and location. But most of them will probably be small since the most weight movements have already been eliminated by the Nguyen-Widrow method to set the initial weights [28]. This leads to more stable networks and a much faster learning process.

4.2.4 Back-Propagation

The neural network is trained with the back-propagation learning algorithm [45]. This learns the weights for a multi-layered feed-forward neural network on training examples. As inputs the sensor information is used and as desired output the steering value representation. With this information the back-propagation algorithm uses gradient descent to minimize an error function which is the squared error between the network's output and the desired output provided within the training examples [40]. The training process has already been described in more detail in section 2.1.
The chosen learning rate influences the convergence time of the learning process. A better way to determine the update amount for the weights is the use of a momentum term (see [7] for more detail). Thereby, the previous update step has influence on the current one. This makes the learning process more stable and accelerates the convergence in shallow regions of the error function. However, the momentum parameter and the learning rate are problem dependent and work better on some problems but lead to no general improvement on others. Furthermore, they have to be chosen before starting the learning process.

Therefore, an adaptive learning algorithms is used here. It is tried to find an appropriate weight update at the beginning and this parameter is updated during the learning process. The learning-rate is modified regarding the observed behaviour of the error function. The weight-step (the change value for the weight updates) depends not only on the learning rate but also on the partial derivation of the error function, as explained in more detail in the next section. Thus, the adaptivity is not determined a priori. This is an advantage of the RPROP learning algorithm [36] which changes the size of the weight-update directly.

4.2.5 RPROP & iRprop

RPROP [36] stands for '**r**esilent **prop**agation' and is an efficient learning algorithm with direct adaptation of the weight step using local gradient information. Each weight has an individual update-value and only this is used for the weight-update. The update-value can be changed during the learning process regarding the weight's local sight on the error function.
The adaption rule reveals that when the partial derivative of the error function changes its sign (which means the last update was too big and the algorithm stepped over a local minimum), the update-value is decreased by a factor. As long as the derivative retains its sign the update-value is slightly increased in order to accelerate the convergence in shallow regions.
The weight-update method itself uses the described update-value and follows a simple rule: If the derivative of the error function is positive (which means an increasing error) the weight will be decreased by the update-value. If the derivative is negative the update-value will be added.
Both update-values and weights are changed every time the whole sample set was presented once to the neural network. This learning by epoch goes on until a predefined minimal error value or a maximal number of epochs is reached.

An advantage of *RPROP* is that for many problems no parameters have to be chosen for optimal or at least nearly optimal convergence time. The method uses a direct adaptation for the size of the weight-update which does not dependent on the derivative value but only on its sign for both learning and adaption. It can easily be implemented and efficiently computed concerning both time and storage. The learning does not take place mainly in some layers or units but is spread equally all over the entire neural network. *RPROP* is robust against the choice of its initial parameters. The number of learning steps is reduced significantly which leads to a much better convergence time.

Here *iRprop* [15], a variation of *RPROP*, is used to train the neural network. This algorithm is only slightly modified in contrast to the one described above. The adjustment of the step-size (the update-value) is still the same but the weight-update works different. The derivative will be set to zero if the sign of the partial derivative of the error function changes. This ensures that in the next step the weight is modified with the reduced step-size but with the current gradient direction.
These variations lead to a significant improvement of learning speed without increasing the complexity of the algorithm. Setting the derivative to zero strongly influences the learning speed. The *iRprop* learning algorithm performs better and converges faster than the original algorithm.

With this algorithm the weights of the neural network were trained on recorded training examples. The trained network was saved for usage in the simulation.

With the `learn` program various neural networks for different input sizes and with varying network architectures (number of units in the hidden layer etc.) can be learned. It is also possible to train two networks for processing stereo camera information, each network for one camera. Since all these parameters are read in from the config file no recompiling is necessary when changing parameters.

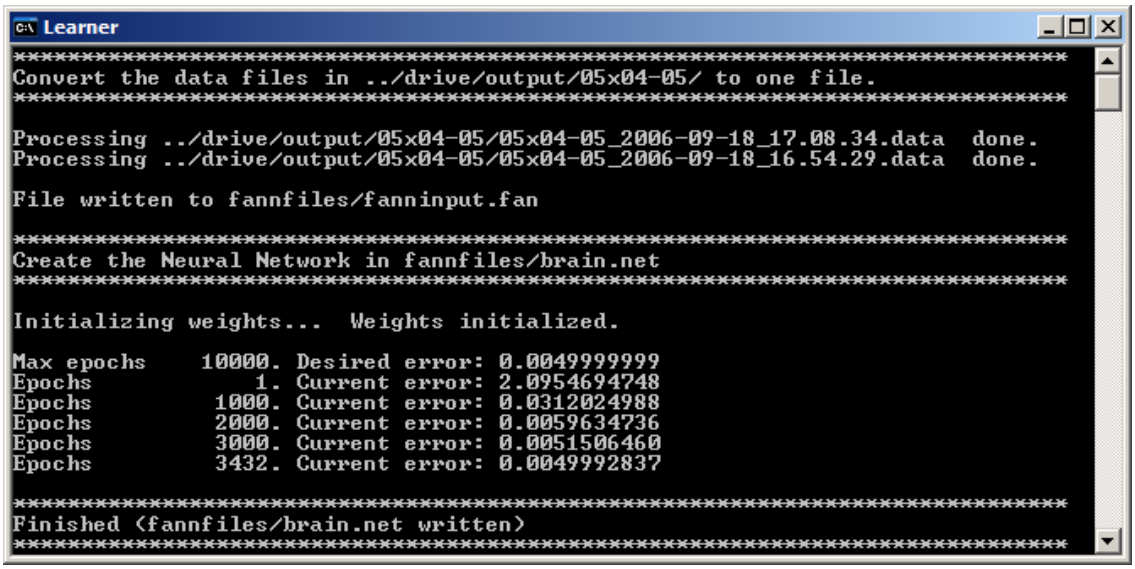

Figure 4.6: The `learn` program

Figure 4.6 shows an execution of the `learn` program. In this example first the two training data files in the specified directory are converted to the *FANN* format and combined in one file (`fanninput.fan`). Afterwards, the neural network is created in consideration of the parameters like the number of inputs, hidden and output units specified in the config file. Then the weights of the network are initialised with the Nguyen-Widrow layer initialisation function and trained on the created file `fanninput.fan`. This is realised with the *iRprop* training algorithm as explained above. The maximal number of epochs is set to 10,000 and the desired error is 0.005. Each 1000 epochs the current error is reported. Here, after 3432 epochs this error is achieved and the learning process stops. The neural network with the trained weights is saved to a file (`brain.txt`) to be used in the simulation.

4.3 The Simulation

In the simulation the pre-trained neural network is loaded to autonomously control the agent for obstacle avoidance or road following. The network which should be used can be defined in the config file. This neural network is loaded in the `Robot` class and used to determine the agent's steering direction.

The input for the neural network is the current view of the agent which is reduced and converted to greyscale. These values are written to an array each frame by the class `AgentView`. The scene is rendered from the agent's point of view in sub-windows depending on the currently used sensors as explained in section 4.2. The brightness of the pixels is stored in an array. The `Robot` class calls a *FANN* method which gives these input values to the neural network and stores the output in an array. There are different possibilities of interpreting the neural network's output. Each output unit represents a steering direction and for each a real value between zero and one is presented. To control the agent a single steering value between zero and one is needed. One possibility is to take just the unit with the maximal activation as steering direction. This maximum can easily be found and can be fast computed. Another possibility is to use the weighted average over all output units and a further to take the mean of the best fitting Gaussian curve of activation over the output units.
All these techniques were implemented and tested. The method used here is to take the mean of the best fitting Gaussian curve near the maximal output unit as the next steering direction. The reasons therefore are given when the results of all methods are compared and discussed in section 5.1.4.

It is also possible to use two neural networks to process the two input images of the stereo cameras and to combine their output values afterwards. Different ways of combining the outputs of these two networks to one steering direction are possible, e.g. to take the maximal value of both outputs and refine the steering direction with the best fitting Gaussian curve over both outputs near this maximum.

After having determined the steering direction the desired movement is checked for collisions with the environment. Here *ColDet*[6], a free 3D collision detection library, was used.

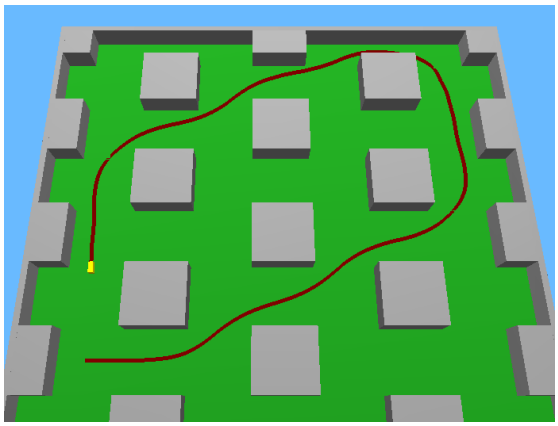

Figure 4.7: Simulation for obstacle avoidance...

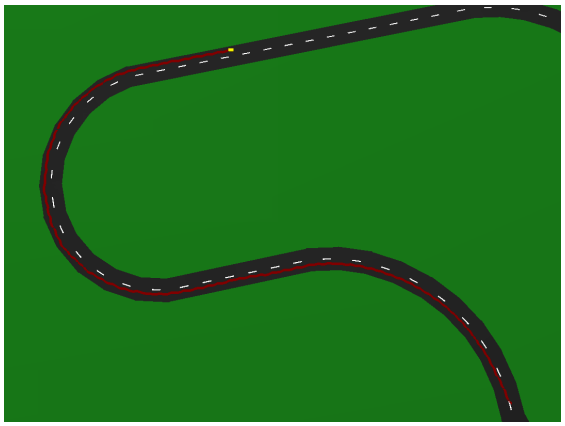

Figure 4.8: ... and road following

The neural network controls the agent autonomously. The path taken by the agent so far can be drawn on the ground (see figures 4.7 and 4.8). This helps to evaluate the neural network's driving ability. For obstacle avoidance it can be determined how well the agent behaves with different sensors, varying view sizes etc. Furthermore, it can be seen how smooth the curves are, how early the agent starts to avoid obstacles and so on. While following a road it can be measured how often the agent leaves the road, if it swerves a lot and how far away it is from the centre of the road.

[6] Available from `http://coldet.sourceforge.net/` as at August 2006

In some cases it is helpful to analyse a trained neural network to gain some further information about the learned features. To visualise the network a weight graph can be created with *fannExplorer*[7], a graphical environment for training, testing and analysing neural networks. Figure 4.9 shows the network topology and the weight graph of a trained fully connected two layer neural network in *fannExplorer*.

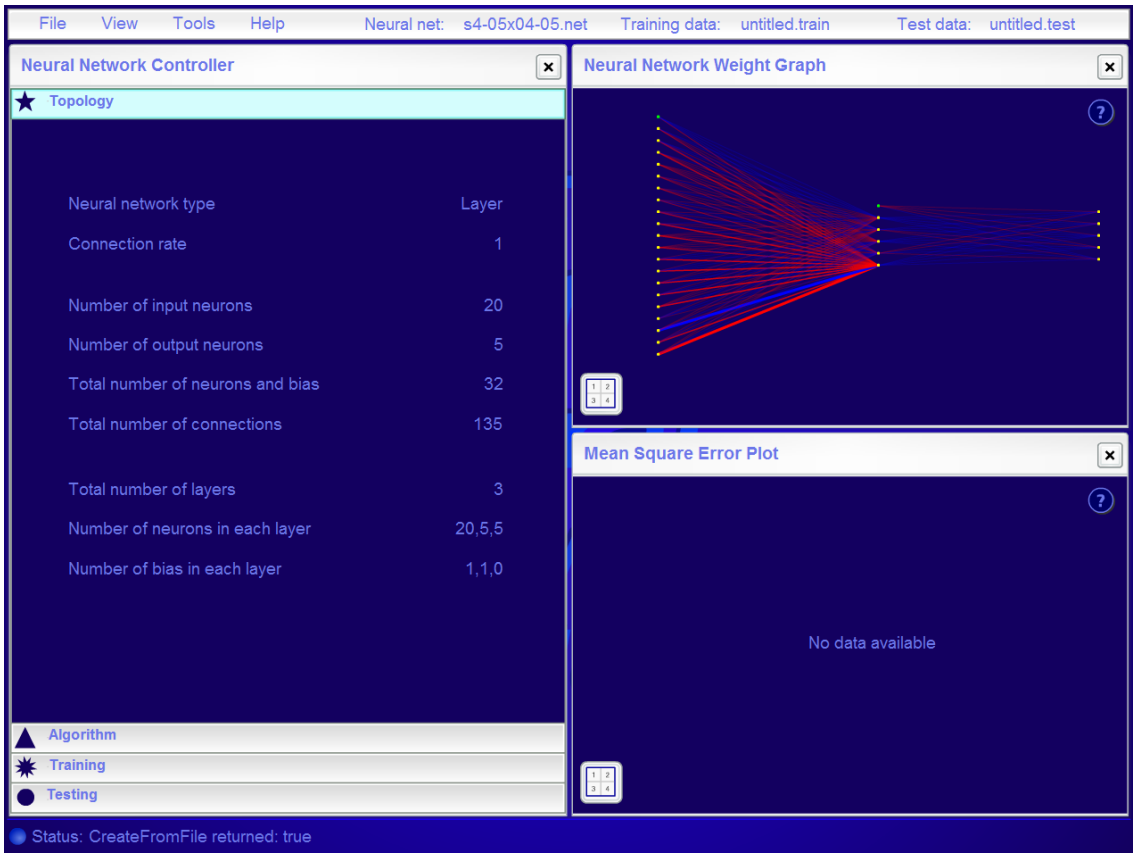

Figure 4.9: Analysing neural networks with *fannExplorer*

[7] Available from `http://www.geocities.com/freegoldbar/indexExplorer.html` as at September 2006

Chapter 5

Results

This section summarises the results obtained within the described investigations for both obstacle avoidance (section 5.1) and road following (section 5.2). The use of neural networks is shown in a virtual environment. Obstacle avoidance worked with different neural networks and input information and the agent was able to follow the road as well. In some cases *fannExplorer* was used to analyse the trained neural networks. Useful information for applications in both real world and virtual environments was gained. The results are presented in the following sections.

5.1 Obstacle Avoidance

With the help of a neural network it was possible to avoid obstacles. The usage of different sensor types for this task as well as different view sizes and varied network architectures was investigated. Furthermore it was tested to combine different sensor types and stereo information.

Therefore, a simple world with varied obstacles in a bounded area was created (see figure 5.1) to gather training data and to simulate the trained network in. Different scenes were used to train and to test the neural network in. It was shown that approximately four minutes of training are enough to learn how to avoid obstacles.

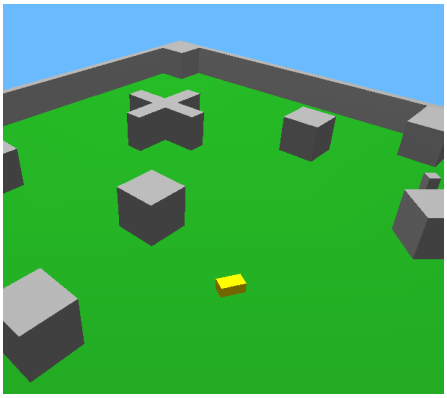

Figure 5.1: Virtual environment

The neural network learned features that are important for obstacle avoidance, e.g. that objects in the distance are not as important as nearer ones. Therefore, the agent only had the information it saw from its own point of view and no map or other representation of its environment that would have allowed to compute an optimal solution. The agent was not told any explicit rules. They would be difficult to name since when creating such rules one has to take many circumstances in account. Instead it only learned from the training examples without being told what to do exactly, and tried to imitate the human driver's behaviour. For example, if the driver turned very early when driving towards an obstacle the trained network would do that as well and the other way around. If the driver behaved very stupid or did nothing the neural network would imitate this behaviour as well.

While recording training data for the neural network the human driver has to take care of the equilibrium between steering to the right and to the left since if the training data is biased to one direction the network's driving will be as well.

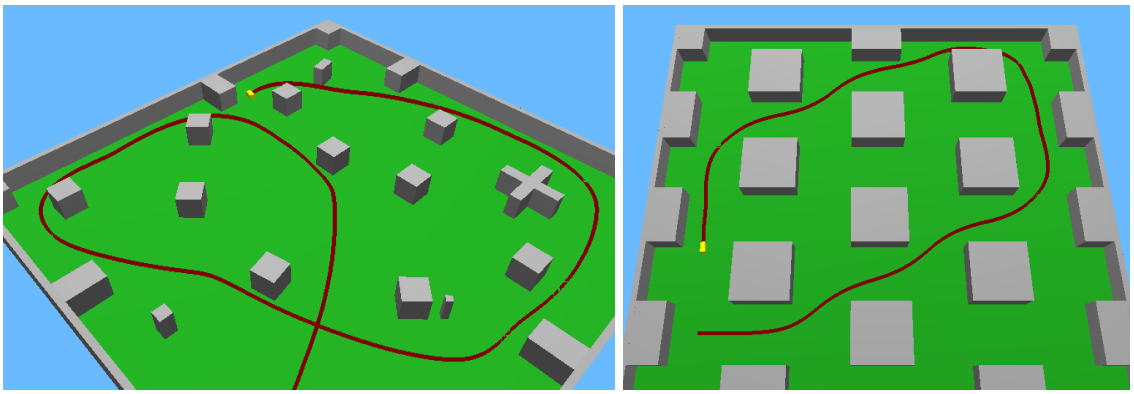

Figure 5.2: Obstacle avoidance on two different maps in a virtual environment

Avoiding obstacles worked not only on the map the agent had been trained on but also in a new environment (see figure 5.2). This shows that the neural network really has learned to avoid obstacles and not only to do exactly what the human driver did on that particular map. This affirms that neural network driving is good especially for arbitrary and unseen environments and therefore a useful application in robotics (e.g. intelligent transport systems) and virtual environments like there are in computer games, for example.

The results with different sensor types and for combining different sensors are presented in sections 5.1.1 and 5.1.2. Diverse neural network architectures and other variations of the learning process are explained in sections 5.1.3 and 5.1.4.

5.1.1 Different Sensor Types

For obstacle avoidance different sensor types were tested. With the chosen sensor input the training examples were recorded and the trained neural network was used to process the steering direction from them.
A single vision camera, stereo cameras, depth information as from a laser range finder and linear cameras were tested with varying view sizes.

With a **single vision camera** obstacle avoidance worked quite well. The agent reacted when driving towards an obstacle with turning, avoided so the obstacle and passed. When driving towards a wall or corner it turned to drive along the wall or away from it again. This worked well even when driving fast (see table 5.2).

Even higher speeds were possible with **depth information** (see figure 5.3, top left). With this input obstacle avoidance worked best as the agent drove very solidly also at high speed (see table 5.2). Depth information like from a laser range finder or the depth buffer is very useful for obstacle avoidance.

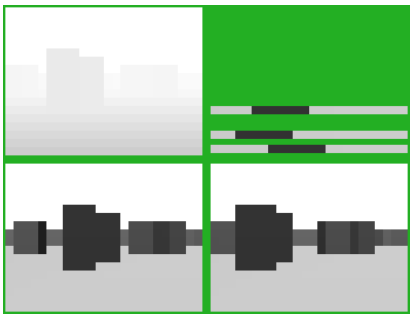

Figure 5.3: Different sensor types

With **stereo vision cameras** (see figure 5.3, bottom) the results were not better than with a single vision camera. Obstacle avoidance was possible but it could have been more stable (see table 5.2). Sometimes, the agent seemed to be unsure which direction to take when avoiding an obstacle especially when driving towards a wall.
Probably, for two stereo cameras more learning is required to gain some additional information out of the two images since the network is bigger and more complicated. The network has not learned that there are two similar images, which input units belong to which sensor image, and which of the pixels in one image corresponds to which pixel in the other image. This would be necessary to

get depth information from the two images which the neural network was not able to. Perhaps, a preprocessing of the images is required to use this information but this would be difficult and computational expensive and it was not goal of these investigations. Another reason for these problems could be that the resolution of the input images was not high enough since images with very high resolution and at best also textures are needed for gathering depth information from images [5].
To combine the inputs of stereo images it was also tried to use two neural networks, one for each camera, and to consolidate the outputs of both. This process is described in more detail in section 5.1.2.

Linear cameras are cameras with only one line as input (see figure 5.3, top right). The line with a high of one third of the vision camera was used since in this line obstacles appear in a reasonable distance to react on them. This one line already contains enough information to avoid obstacles. With a single linear camera obstacle avoidance already worked with a view width of three pixels, but not very well. Much better results were obtained with a width of 10 pixels and more, the more the better. Table 5.1 shows the collisions with the environment that occurred in a certain time while driving with a linear camera as sensor input and different camera widths with the same high speed for each width.

camera width	3	6	16	24	30
collisions	10	9	7	5	4

Table 5.1: Collisions with a linear camera and different camera widths

Furthermore, driving with a linear camera led to more immediate turns since the sight of such a camera is obviously not that far. Because of this limited view the agent only driving with linear camera information had problems when it was too close to an obstacle, especially when this was a wall. Nevertheless, it was possible to avoid obstacles with only that one line of information. And of course the learning was faster since there were only few input units. However, it was only slightly faster.

With stereo linear cameras the results were quite similar (see table 5.2). For two linear cameras the view width even has to be bigger for successful obstacle avoidance. It only worked above a minimal width, e.g. not with 16 or fewer pixels but with 20 or more. With too few pixels the agent was sometimes not sure which direction to take when an obstacle or especially when a wall was straight ahead. Furthermore the turns were not sharp which led to slower possible speeds.
In summary stereo linear cameras were not as good as a single linear camera. Two linear camera inputs seem to contradict each other. The agent was not able to learn features from stereo vision, the second linear camera only disturbed the steering. Nevertheless, obstacle avoidance was possible with two linear cameras as well.

Recapitulating, linear cameras already provide enough information to avoid obstacles. It works well above a certain minimal image width. The agent does not behave as cleverly as with the whole image as it has much less information. Such a simple network (with that few input units) can already learn obstacle avoidance. And of course learning is much faster since there are much less input units.

Altogether, depth information like from a laser range finder worked best for obstacle avoidance. This information is very useful for that task. With a vision camera it worked quite well, too, but a linear camera is already enough to avoid obstacles properly above a necessary camera width. The stereo information did not lead to better results since the neural networks did not seem to be able of gaining new information out of the stereo images.

Table 5.2 shows the collisions with obstacles that occurred with different sensor types in a certain time. For each sensor the driving speed was the same high value. The number of collisions can

be seen as a measurement of stability and obstacle avoidance quality of a neural network. That there occur so many collisions in some cases originates from the high driving speed.

sensor \ view size	3×2	24×18
single camera	2	2
depth information	0	1
stereo cameras	3	2
linear camera	10	4
linear stereo camera	9	5
depth info + single cam.	4	0
depth info + linear cam.	4	3

Table 5.2: Collisions with different sensors

5.1.2 Combining Information

As described above neural networks had problems to combine **stereo** information. They did not learn to get depth information from stereo images like a human would do. Instead of providing advantages the network was more complicated, needed longer time to learn and did not lead to better results. The neural network could not determine which input was from which picture.
It was tried to go about this problem with **two neural networks** for stereo cameras, one for each camera image. The networks were trained with only one of the stereo images each. Afterwards, they were used to process the same stereo image (left or right, respectively) and to compute a steering direction from that. The two outputs were combined to a single steering value either by taking the maximum of both network's outputs or by taking the average between the maximum of each networks or by other methods. However, in all cases the agent was biased to drive to one direction or mainly straight ahead in the simulation. The result of using stereo cameras for obstacle avoidance was not satisfying.
With two linear cameras the result was quite similar. The agent was mostly biased to one direction and made many mistakes, e.g. driving against a wall or an obstacle.
Altogether, this method did not work very well, at least not better than one neural network for both inputs or only a single camera. Perhaps the results would be better with a pre-processing of the stereo images that extracts the depth information from them. However, this is not part of this project. Thus, the method using two neural networks for stereo images was not investigated any further.

Another way to combine stereo information instead of using a fully connected neural network might be to use a neural network only providing a carefully **selected subset of connections**. Thus, a network in which the inputs of each stereo image are connected to only one half of the hidden units can be created. Until this first processing of each image the stereo information is not combined in the next layer which is fully connected to the first hidden layer. With this method, the two stereo images can be reduced to the activation of much less units and can be combined afterwards. However, the used neural network library[1] does not allow such kind of influence to the neural network architecture. Therefore, investigating variations like this is future work.

To obtain better driving results in the purpose of obstacle avoidance **combining different sensors** was tried. With the input from two sensors more information is available to rely on in order to avoid obstacles. This was tried with the combination of a normal camera and depth information and with the combination of a linear camera and depth information. Combining linear cameras with normal cameras makes no sense since the normal camera already contains the line of information the linear camera provides. Neither does the combination of depth information and stereo images make sense since depth information with normal cameras were tested anyway and stereo

[1] *FANN*, available from `http://leenissen.dk/fann/`

cameras were shown to work no better than a single camera (see section 5.1.1).
Combining the normal camera with the depth information resulted in a slightly better driving behaviour than with each information alone when the view size was big enough. The analysis of the network showed that the agent relies on both inputs and not only on one, e.g. only on the depth information as one could guess since depth information worked best for road following. With depth information and a linear camera the agent behaved better than with only the linear camera. Again, the network analysis showed that both inputs were used.

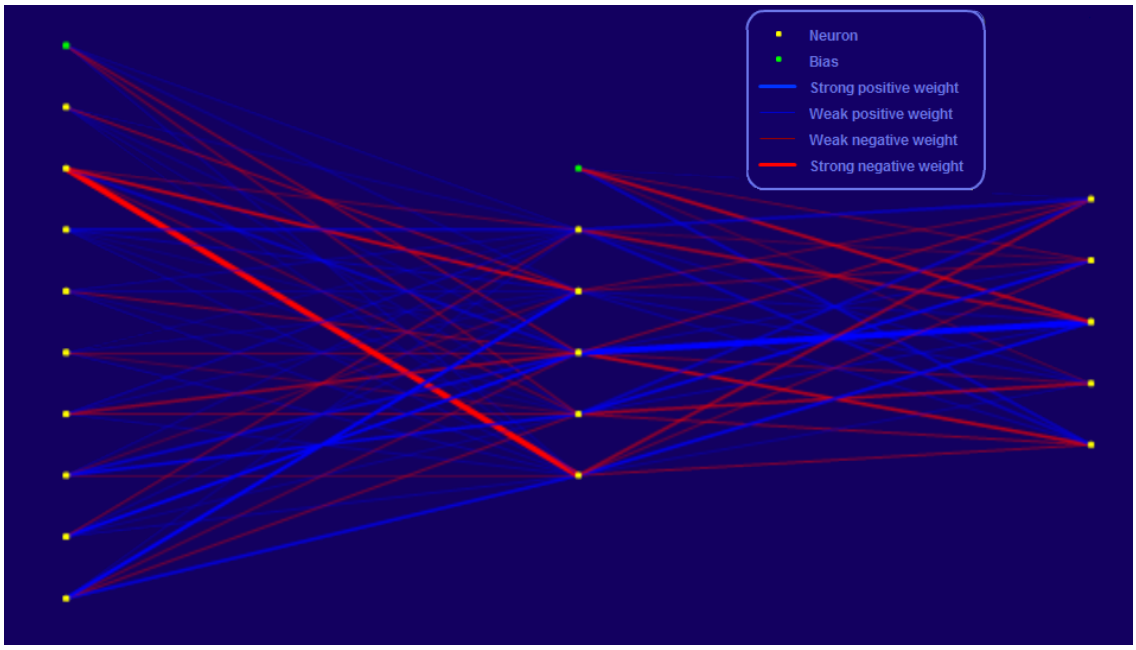

Figure 5.4: Weight graph of neural network trained on depth information and a linear camera

Figure 5.4 shows the weight graph of a neural network trained on both depth information and the input of a linear camera. The network consists of nine input units (left), five hidden units (middle) and five output units (right). Blue lines represent positive weights, red ones negative weights. The thicker the lines the stronger the weights. The lower six points on the left are the input units for the depth information, the three above those of the linear camera. Obviously, not only information of one sensor is used to determine the steering direction since for both sensor types strong weights exist. The weight graph was created with *fannExplorer*.

Altogether, it was shown that two neural networks lead to no better result when processing stereo images but combining different sensor types with a neural network can result in better driving behaviour than with each of the sensors alone.

5.1.3 Neural Network Architectures

While testing the neural networks it turned out that the quality of the trained network differed when training with the same data and the same parameters again. Especially for big view sizes the networks have been sometimes very good and sometimes quite bad with the same training. This became much better when **initialisation of the network weights** was used. With this technique not only the results were better and much more stable but also the learning was faster. first random values were used to initialise the weights. With initial weights between -10 and 10 it worked well, better than with other tested values, e.g. between -100 and 100 or between -1 and 1. Even better results were obtained with the Nguyen-Widrow layer initialisation function [25]. This worked very well and resulted in stable networks and faster learning than without weight initialisation or with taking random weights.

Additionally, **different network topologies** were tried. The neural network's architecture was varied, especially the hidden layer. More and less hidden units have been tested as well as no hidden layer at all.
Normally, approximately four to seven hidden units were used, depending on the number of output units. Using only three hidden units there was not much difference to using four or five hidden units. Using only one hidden unit obstacle avoidance still worked, but not as well as with more hidden units. The turns were not that sharp and so the speed had to be slower. If there is only one hidden unit this unit will represent the steering direction. Thus, only one real value representing the steering direction is enough, but using more output units allows smoother driving and more accurate learning. More hidden units than usual (for example 15 or more) resulted in not much difference regarding driving but the learning process took longer.
Table 5.3 shows the training times for 10,000 epochs (see section 4.2.5) for different number of hidden units. Values for a neural network with no hidden layer and for networks with up to 15 hidden units are given. The neural networks were trained on depth information with a three times two sensor view (i.e. six input units) and five output units on a 3.40GHz PC).

hidden units	15	10	7	5	3	1	0
learning time	14.8s	9.7s	8.9s	8.0s	6.9s	5.4s	5.1s

Table 5.3: Training times for different number of hidden units

With no hidden layer at all obstacle avoidance worked well for small view sizes. The steering direction depends directly on the input values. However, for bigger view sizes it did not work very well, e.g. with a width of six pixels the driving was good, with 16 it was not that good anymore and with a width of 30 pixels obstacle avoidance did not work at all without any hidden layer. Thus, using no hidden layer is only appropriate for very small neural networks or rather few input units.

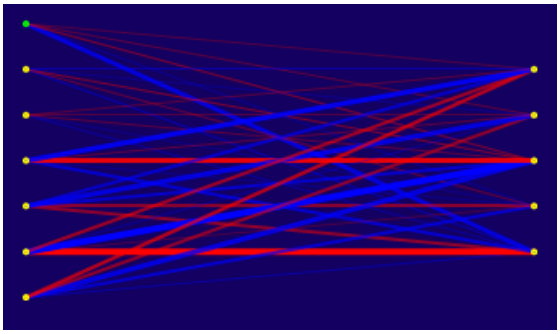

Figure 5.5: Weight graph of neural network with no hidden layer for depth information

Figure 5.5 shows the weight graph of a neural network trained on depth information with a view size of three times two pixels and no hidden layer. The units on the left are the input units, those on the right the output units. The neural network relies mainly on the lower line of pixels (the units on the left bottom) since obstacles occurring there are closer and more likely to react on them. Obstacles on the right side mainly activate output units representing steering directions to the left and vice versa.

Another way to vary the neural network topology is to use a network consisting only of a carefully **selected subset of connections** instead of a fully connected one. Thus, two different sensor inputs cannot be combined until the second layer, for example, or features can be extracted by first combining pixels next to each other or on certain positions. However, *FANN* does not support such a kind of influence on the neural network architecture. Therefore, those investigations are future work.

Finally, the **connection rate** of the neural network was varied. The connection rate is the percentage of connections which exist between each layer and the next one in comparison to all possible connections between these layers. Normally, the connection rate was set to one which means that the network is fully connected, thus every unit is connected to every unit in the following layer. If the connection rate is less than one (it has to be above zero) learning is faster since less weights need to be learned. It was investigated how much the connection rate can be reduced without loosing too much quality.
Different connection rates were tested; with a connection rate of 1.0 obstacle avoidance worked

well as described above. Using a connection rate of 0.5 it still worked but the curves were longer and not as sharp as before. Nevertheless, the result was satisfactory. Testing this connection rate with a very small view size (e.g. three times two pixels) the result was not as good as before but remained acceptably. With a connection rate of 0.2 there were still observable tries to avoid obstacles but the network was not very robust. The agent made many mistakes and the driving behaviour was not satisfactory anymore. Using only a small view size nearly no turning and therefore no obstacle avoidance at all could be observed. With 0.1 as connection rate nearly no obstacle avoidance was observable but the agent changed the direction only slightly which was not enough to avoid all obstacles. Some examples with neural networks using a sensor view of three times two pixels, depth information, and different connection rates is shown in figure 5.6. Figure 5.7 shows the weight graph of a neural network with a connection rate of 0.2. Only 20% of the possible connections between the units exist.

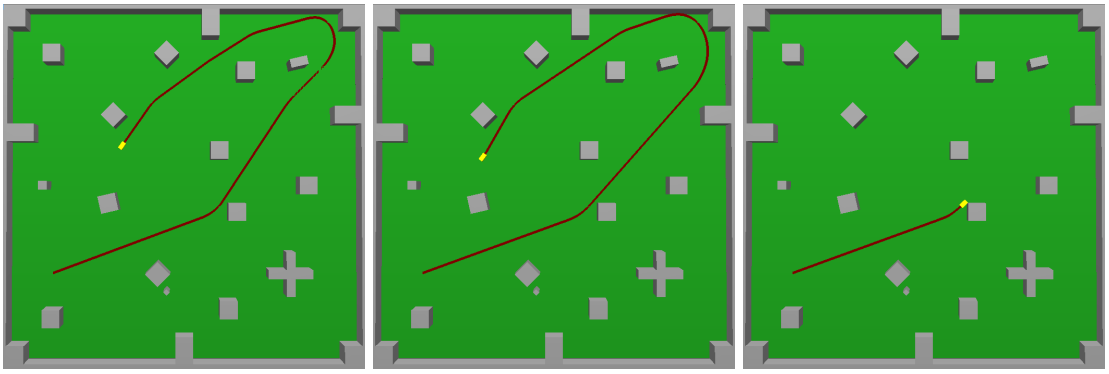

Figure 5.6: Driving behaviour of neural networks with a connection rate of 1.0, 0.5, and 0.2, respectively (from left to right)

The training times for 5,000 epochs (see section 4.2.5) for the different connection rates are given in table 5.4 (here a five times four sensor view (i.e. 20 input units) and four hidden units were used on a 3.40GHz PC).

connection rate	training time
1.0	2914 ms
0.5	2473 ms
0.2	2055 ms

Table 5.4: Training times for different connection rates

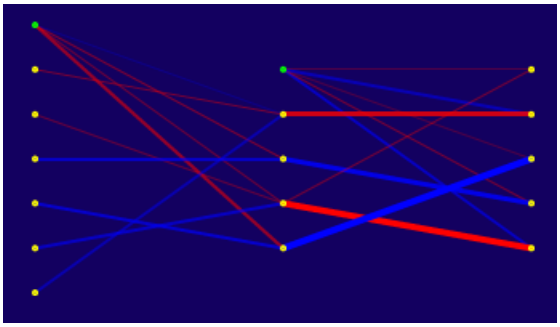

Figure 5.7: Weight graph of neural network with a connection rate of 0.2

Altogether, 50% of the connections can be enough for successful obstacle avoidance if the network or rather the view size is big enough. Below that value the driving behaviour is not satisfactory anymore, especially for small networks and input sizes. Therefore, a connection rate of 1.0 is most reasonable here because on the one hand the gain of time during the network's training by using a lower connection rate is rather small and not necessary and on the other hand a connection rate of 1.0 works better and leads to more reliable networks.

5.1.4 Further Variations

Other tested variations include different view sizes for neural network input, the use of the last steering output as further input and different methods to compute the steering direction out of the activation of the network's output units.

The **view size** (the size of the view in pixels which a sensor produces) was varied while testing to investigate which differences occur in the driving behaviour and which minimum of pixels is needed to avoid obstacles. It was shown that only a few pixels are already enough. With six pixels of depth information (a three times two pixels view) very stable obstacle avoidance was possible and since the resulting network was very small learning was quite fast. Using a single vision camera with the same view size worked as well but not as solidly as with the depth information. For linear cameras good results were obtained with a width of ten pixels and more. It was also possible with three pixels but not very stable as described earlier. With more pixels per image (up to 30 times 24 were tested here) the steering became more accurate until a threshold was reached above which there were no changes anymore. However, as explained, a very small amount of pixels is already enough to avoid obstacles. The neural network learned features with such little information from the sensor image so that a human would have difficulties to react on in order to avoid obstacles.

The **last steering value** of the neural network was used as further input to give the network information about the current turning direction and amount of the vehicle. This could be helpful to avoid driving into a corner where the wall on the left and the wall on the right influence the agent to drive to the right or to the left, respectively. Without the information of the last steering output the agent sometimes was likely to drive directly in the corner and against the wall while alternating from the left to the right and vice versa. If the agent relied on the last steering output this alternation could be avoided and the agent would be able to decide one driving direction and to avoid the corner.
However, when the last steering direction was used as further input for the neural network the agent drove in circles and mostly did not avoid obstacles well. That seemed to be the case because the neural network learned to rely mainly on the last steering but hardly on the sensor information. Especially with no hidden layer the agent drove in a circle when it started to turn the first time. It avoided the first obstacle but kept turning to the same side afterwards. This was not much better when hidden units were used. Using weight initialisation improved the behaviour a bit but the agent was still not avoiding obstacles but driving in circles. With the last steering value the learning converged faster and the errors were smaller which is also an indication for the supposition that the neural network learned mainly to rely on the last steering value.

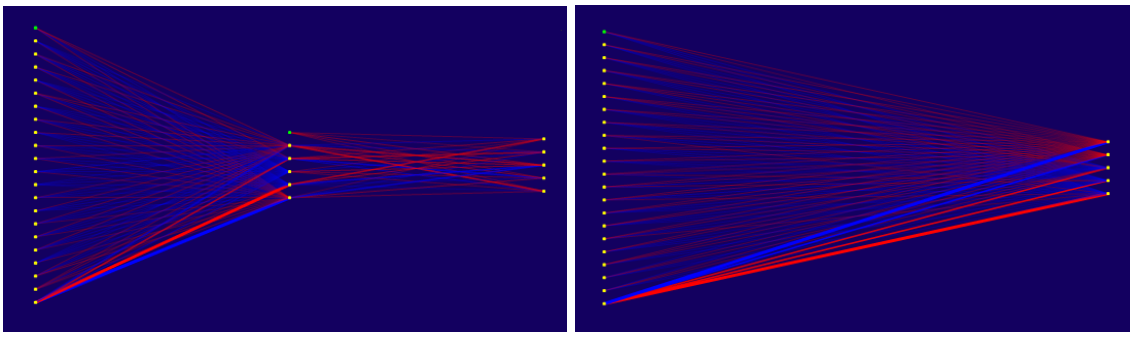

Figure 5.8: Networks using the last steering value (five and zero hidden units)

To verify this assumption the trained neural network was analysed with *fannExplorer*. In figure 5.8 the weight graphs of trained networks for depth information with a view size of five times four pixels using the last steering value as further input with five and without hidden units, respectively, is shown. It is obvious that the input unit with the last steering input (the unit at the very left bottom) has much higher absolute weights than the other inputs (the units on the left). The current steering direction depended mainly on the last steering value and not on the sensor input. Therefore, the last steering value seemed not to be useful and was not used here.

While gathering **training data** for the neural network the human driver had to take care of the equilibrium between steering to the right and steering to the left. When there was a strong disequilibrium between the two directions the agent was biased as well and was more likely to

drive to one direction than to the other. This could be corrected by recording more training data where the driver steered more often to the other direction and then training the neural network again on both the old and the new data.

Different output interpretations of the neural network were tested. Since the neural network has several units of which each has a certain activation as output it is not trivial to get one steering direction out of this information. During the training the desired output was set to a hill of activation representing a Gaussian curve around the current steering direction. This was done to allow more accurate steering and more diverse driving behaviour than with only one output unit representing the driving direction.
To use the neural network's output activation to obtain one steering direction different methods were tested. First, simply that steering direction represented by the output unit with the *maximal activation* was used as current steering direction. This worked very well for the purpose of obstacle avoidance but the agent's driving was not smooth. It drove straight ahead until an obstacle was in the way and then turned with a discrete radius before suddenly driving straight again. This did not look very naturally.

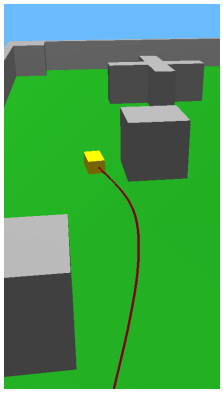

Figure 5.9: Obstacle avoidance

To get a smoother driving behaviour it was tried to use the weighted average over the output units and the best fitting Gaussian curve, too. Both did not result in good obstacle avoidance since the curves were not sharp and the agent drove nearly straight ahead most of the time. A possible reason for that is given in the following; since all units have a certain degree of activation the weighted average of them is very often near the centre. The same will hold for the best fitting Gaussian curve if the activation is not already formed as a Gaussian curve. Especially when it was unclear to which direction the agent should drive in order to avoid an obstacle both weighted average and best fitting Gaussian curve resulted in a steering direction straight ahead. Furthermore, the calculation of the best fitting Gaussian curve over all output units is computational expensive.
The best results were obtained by using the mean of the best fitting Gaussian curve near the maximal value as representation for the next steering direction. With this method the agent drove smoother than with only the maximum as steering direction since this method avoided the disadvantages of the weighted average and the best fitting Gaussian curve over all output units as described above. The agent drove real curves, was able to avoid obstacles very well, and steered very smoothly. Smaller curves were possible since the steering was not discrete anymore but continuous now. Additionally, the calculation is quite fast.
It has been shown that with this method the activation of several output units can be translated into a single steering direction effectively and satisfyingly for the purpose of obstacle avoidance. It was quick to compute, worked well and resulted in a really smooth driving behaviour.

5.2 Road Following

In this section the results of using an neural network for road following are described. The goal was to train the neural network to follow the road as accurately as possible, i.e. stay in the centre of the road or a lane. The agent should turn when the road is curved and drive straight when there are no curves on the road. Furthermore, the neural network should be able to correct from swerving if the agent left the road.

Therefore, a map with a curved road was used to train and to test the neural network. The agent had no prior information about the road geometry like a map or another description. It had only the information it saw from its point of view to steer in order to stay on the road. Thus, a human or a real robot with a vision camera trying to follow the road would be in the same situation. Especially in the field of virtual environments other techniques like to compute optimal steering values from a road geometry representation in a map for example are possible. However, using only the visible information results in a more natural behaviour. Therefore, deploying neural networks for road following is appropriate to both virtual environments such as in computer games and the real world.

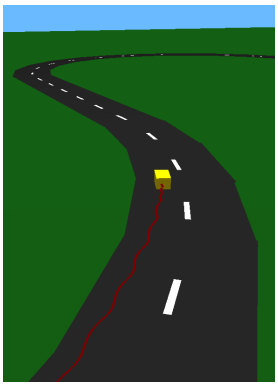

Figure 5.10: Road following

A neural network trained on the data recorded by a human driver following the road was able to follow the road. After having trained the neural network the agent stayed mostly on the road and therefore followed it. The agent imitated the human's driving behaviour, e.g. drove in the middle of the road when the driver did so and mainly on one lane when the driver did. Moreover, the agent could follow curved roads (see figure 5.11). Thereby, the white stripes in the middle of the road did not seem to be important since the agent followed the road regardless the presence of such white stripes.
However, the driving behaviour was not smooth, the agent was swerving from the centre of the road and sometimes even left the road (see figure 5.12).

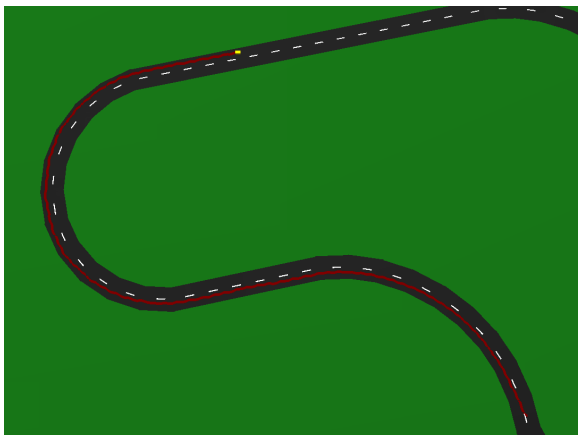

Figure 5.11: The agent following the road

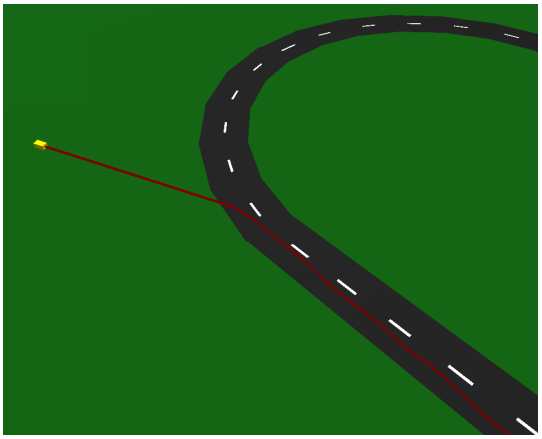

Figure 5.12: The agent leaving the road

This unstable driving behaviour was improved by the method to transform the images and to compute the appropriate steering direction for those situations to get more diversity in the training data as described in section 4.2.1. With the resultant additional information the neural network was able to learn how to correct mistakes. Hence, when leaving the road it knew how to drive back to the road. Then admittedly the agent did not drive only on the lane of the road it was trained on but on the whole road. Furthermore, the agent still swerved on the road but now with higher amplitude over the whole road. Both may originate from the transformation of the images

adding information to the training data in which the road is seen from different position and angles. It is possible that the parameters for the calculation of the appropriate steering direction to the transformed images, e.g. the look-ahead distance, were not chosen perfectly adequate. Additionally, while recording data the performance was decreased radically with this method since instead of one image 17 images had to be rendered.

Altogether, the agent learned to follow the road only using the information it could see from its own point of view. To improve its driving behaviour the method to transform the images is two edged: on the one hand, the neural network can be shown how to correct mistakes and how to drive back to the centre of the road. On the other hand, driving while recording data is a lot slower and the driving is not much smoother afterwards.

Nevertheless, neural networks have been shown to be useful for road following in a virtual environment, especially if a natural driving behaviour is desired. Such a system can be used in computer games, for example. Autonomous agents can be controlled simply without telling them any complex rules or giving them a detailed representation of their environment like a map. An agent can learn how to behave just from the visual information it gets from its point of view.

Chapter 6

Conclusion

The purpose of this project was to investigate the use of artificial neural networks for autonomous driving. Therefore, the processing of inputs from various sensor types like a single camera, stereo vision, depth information (like that from a laser range finder) and linear cameras was tested for obstacle avoidance and road following. For that purpose a human driver can gather training data in a virtual environment. This recorded data is used to train a neural network which then generates the steering values for an autonomous mobile agent as output.

It was shown that neural networks are an appropriate method to gain a steering direction out of sensor information in order to drive a vehicle and to avoid obstacles or to follow the road.

For the purpose of obstacle avoidance, depth information like from a laser range finder is most useful. A vision camera also works very well, and a linear camera gives enough information for that task as long as the view size is above a certain threshold. Stereo cameras do not lead to better results since they require more training and the emerging network is more complicated. Combining different sensors can help to improve the driving quality. In contrast, using two neural networks to process stereo images results in a defective driving behaviour.
Initialisation of the neural network's weights is essential for an accurately driving agent. Indeed, the connection rate can be reduced to 50% in most cases neither providing any disadvantage nor a big advantage. Furthermore, it was shown that only a very small view size is needed to avoid obstacles. The representation of the steering direction with several output units works well. Thereby, using the best fitting Gaussian curve near the maximal activated unit to get the steering direction out of the activation of the output units is the most successful method. It permits smooth and stable obstacle avoidance.
Neural networks are particularly useful for obstacle avoidance. Especially the results concerning the ability of an autonomous agent to avoid obstacles with different sensor information can be helpful for both agents in virtual environments and robots in the real world. Combining information from different sensor types with neural networks can be useful for obstacle avoidance and can improve the driving behaviour. The results about the output interpretation are interesting and the chosen Gaussian curve was shown to work very well.

It was shown that a neural network controlled agent is capable of road following even after a very short time of training. The agent can be trained to follow one lane or the whole road. In the latter case it was proven that the agent does not rely on the white stripes in the middle of the road.
Nevertheless, the agent is swerving while following the road and sometimes even leaves it. Using image transformations the neural network is presented training examples that show how to correct mistakes. Therefore, the human driver does not have to leave the centre of the road during the training since the images are shifted and rotated and the appropriate steering direction is computed. However, this makes the recording process computational expensive. With this method the agent drives on the whole road and is still swerving but not so likely to leave the road anymore. Thus, the technique to transform the images helps to follow the road.

These investigations proved that neural networks are useful means to control autonomous mobile agents. There are various possible applications. In a virtual environment neural networks can help to achieve a more realistic behaviour of driving agents. For example, the computer controlled opponents in a computer game could be controlled by a neural network. Thus, their behaviour becomes more natural since they would only see what a human driver would see from their point of view. Moreover, no complex rule based algorithms are necessary to control the agents' steering. The obstacle avoidance and road following abilities can also be useful for real autonomous driving agents as they exist in the industrial field, e.g. intelligent transport systems in factories or repositories. Furthermore, neural networks can help to improve driving control systems that warn the driver when he is about to leave the road. Thereby, different sensor types are deployed and neural networks were shown to be capable of processing diverse information.

Altogether, neural networks are an appropriate method to teach autonomous agents obstacle avoidance and road following. They are capable of combining different sensor information and of producing an adequate steering output. The results of this project can help to optimise the sensor types, the view sizes, the neural network topology, the output interpretation and further parameters for autonomous neural network driving tasks.

Chapter 7

Future Work

Even though many sensor types were already tested in the investigations of this project, more different sensors like infrared distance sensors or sensor systems with structured light could be tested for their use in obstacle avoidance. For road following different road types like roads with several lanes can be used to train a neural network on. To avoid a biased training set the mirrored image and the appropriate steering direction for each image could be used as well.

Moreover, the environment could be more complex. Although a flat world was sufficient for the investigations of this project the influence of an environment with hills would be interesting. Textures could be added to the world as well as some objects like trees. To gain a more realistic look the use of a game engine (e.g. Ogre or Irrlicht) instead of `OpenGL` could help.

Furthermore, other network topologies could be tested, especially to combine the stereo information. Further variations to the network topologies than those described earlier were not tested in this project since the used neural network framework does not allow the desired kind of influence on the neural network topology. However, with another framework neural networks with only a carefully selected subset of connections existing can be possible. Therewith, each stereo image can be connected to one half of the hidden units and the stereo information is split up to the second hidden layer, for example. It could be tested to combine the two images, similarly to how it happens in the human's eyes and brain. It could be possible to extract some features first and combine this information afterwards in order to get an appropriate steering direction for the current situation. Therefore, the pixels near each other or those on one line can be combined first to get some further knowledge about the image on a higher level than just the pixel values. This information can be used additionally to train a neural network.

Moreover, a neural network could be used to control not only the steering direction but also the velocity of a virtual vehicle. Another idea is to use two networks to control an agent, one for the steering direction and one for the velocity.

It is also possible to test the driving behaviour of multiple agents. Therefore, the views of the agents should be rendered off-screen. Several agents could be trained to react on each other, to avoid each other or to follow each other.

Another improvement for the system could be the use of a steering wheel as control input for the human driver. Therewith, the steering values recorded in the training data could be more accurate. It could be investigated whether this leads to a better obstacle avoidance or road following behaviour.
Additionally, a further idea would be to use especially the road following for a computer racing game. However, for that purpose several adjustments to the existing system would be necessary. Furthermore, a neural network could be trained to detect and to traverse intersections.

Bibliography

[1] M.J. Aitkenhead and A.J.S. McDonald. A neural network based obstacle-navigation animat in a virtual enviroment. *Engineering Applications of Artifical Intelligence*, 15:229–239, 2002.

[2] S. Baluja. Evolution of an artificial neural network based autonomous land vehicle controller. In *IEEE Transactions on Systems, Man and Cybernetics, Part B*, volume 26, issue 3, pages 450–463. IEEE, 1996.

[3] A. Barto. Connectionist learning for control: An overview. In W. T. Miller, R. S. Sutton, and P. J. Werbos, editors, *Neural Networks for Control*, pages 38+. MIT Press, 1989.

[4] P. Batavia, D.A. Pomerleau, and C. Thorpe. Applying advanced learning algorithms to ALVINN. Technical Report CMU-RI-TR-96-31, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, October 1996.

[5] S. Birchfield and C. Tomasi. Depth discontinuities by pixel-to-pixel stereo. In *ICCV*, pages 1073–1080, 1998.

[6] G.W. Cottrell. Extracting features from faces using compression networks: Face identity emotion and gender recognition using holons. In *Connectionist Models: Proceedings of the 1990 Summer School*, pages 328–337, 1990.

[7] G. Dreyfus. *Neural Networks*. Springer-Verlag Berlin Heidelberg, 2005.

[8] Meng Joo Er and Chang Deng. An intelligent robotic system based on neural-fuzzy approach. In *Proceedings of the 7th International Conference on Control, Automation, Robotics and Vision, 2002*, volume 2, pages 619–624, December 2002.

[9] E. Eskafi and D. Khorramabadi. Smartpath user's manual. Technical report, University of California-Berkeley, December 1990.

[10] S.E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical report, Computer Science, Carnegie-Mellon University, 1988.

[11] S.E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532. Morgan Kaufmann, San Mateo, 1990.

[12] Daniel Flower. Simulated visual perception for autonomous agents, 2005. Project, Department of Computer Science, University of Auckland, `http://www.cs.auckland.ac.nz/~burkhard/Reports/2005_S1_DanielFlower.pdf`.

[13] Daniel Flower, Burkhard C. Wünsche, and Hans W. Guesgen. Simulated visual perception-based control for autonomous mobile agents. In *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2006*, pages 725–730, Menlo Park, Calif., 2006. AAAI Press.

[14] C. Hansch and A. Leo. Exploring QSAR-fundamentals and applications in chemistry and biology. *American Chemical Society*, 1995.

[15] C. Igel and M. Hüsken. Improving the Rprop learning algorithm, 2000.

[16] L.D. Jackel, H.P. Graf, W. Hubbard, J.S. Denker, D. Henderson, and I. Guyon. An application of neural net chips: Handwritten digit recognition. In *Proceedings of the 1988 IEEE International Conference on Neural Networks*, volume 2, pages 107–115, 1988.

[17] D.Y. Jeong, S.J. Park, S.H. Han, M.H. Lee, and Takanori Shibata. A study on neural networks for vision-based road following of autonomous vehicle. In *Proceedings of the 2001 IEEE International Symposium on Industrial Electronics*, volume 3, pages 1609–1614, 2001.

[18] T. Jochem and D.A. Pomerleau. Vision-based neural network road and intersection detection. In Martial H. Hebert, Charles Thorpe, and Anthony Stentz, editors, *Intelligent Unmanned Ground Vehicles*. Kluwer Academic Publishers, 1997.

[19] T. Jochem, D.A. Pomerleau, and C. Thorpe. MANIAC: A next generation neurally based autonomous road follower. In *Proceedings of the International Conference on Intelligent Autonomous Systems*, volume 3, 1993.

[20] T. Jochem, D.A. Pomerleau, and C. Thorpe. Vision-based neural network road and intersection detection and traversal. In *Proceedings of the IEEE Conference on Intelligent Robots and Systems '95*, volume 3, pages 344–349, August 1995.

[21] T. Jochem, D.A. Pomerleau, and C. Thorpe. Vision based intersection navigation. In *Proceedings of the 1996 IEEE Symposium on Intelligent Vehicles*, pages 391–396, September 1996.

[22] T. Jochem, D.A. Pomerleau, and Chuck Thorpe. Vision guided lane transition. In *IEEE Symposium on Intelligent Vehicles*, pages 30–35, September 1995.

[23] L.P. Kaelbling, M.L. Littman, and A.P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[24] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.J. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[25] D. Nguyen and B. Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *IEEE International Joint Conference on Neural Networks*, volume 3, pages 21–26, 1999.

[26] S. Nissen. Implementation of a fast artificial neural network library (fann). Technical report, Department of Computer Science, University of Copenhagen (DIKU), October 2003.

[27] Se-Young Oh, Jeong-Hoon Lee, and Doo-Hyun Choi. A new reinforcement learning vehicle control architecture for vision-based road following. In *IEEE Transactions on Vehicular Technology*, volume 49, issue 3, pages 997–1005. IEEE, May 2000.

[28] A. Pavelka and A. Prochzka. Algorithms for Initialization of Neural Network Weights. In *Sbornk prspevku 12. rocnku konference MATLAB 2004*, volume 2, pages 453–459, 2004.

[29] T.F. Pawlicki, Dan-Shyang Lee, J.J. Hull, and S.N. Srihari. Neural network models and their application to handwritten digit recognition. In *Proceedings of the 1988 IEEE International Conference on Neural Networks*, volume 2, pages 63–70, 1988.

[30] D.A. Pomerleau. ALVINN: An autonomous land vehicle in a neural network. *Advances in neural information processing systems 1*, pages 305–313, 1989.

[31] D.A. Pomerleau. Progress in neural network-based vision for autonomous robot driving. In *Proceedings of the 1992 Intelligent Vehicles Conference*, pages 391–396, June 1992.

[32] D.A. Pomerleau. *Neural network perception for mobile robot guidance.* Kluwer Academic Publishing, 1993.

[33] D.A. Pomerleau. Neural networks for intelligent vehicles. In *Proceedings of the 1993 Intelligent Vehicles Conference*, pages 19–24, July 1993.

[34] D.A. Pomerleau. Neural network vision for robot driving. In *The Handbook of Brain Theory and Neural Networks.* M. Arbib, 1995.

[35] D.A. Pomerleau and T. Jochem. Rapidly adapting machine vision for automated vehicle steering. *IEEE Expert: Special Issue on Intelligent System and their Applications*, 11(2):19–27, April 1996. See also IEEE Intelligent Systems.

[36] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. of the IEEE Intl. Conf. on Neural Networks*, pages 586–591, San Francisco, CA, 1993.

[37] M. Rosenblum. Neurons that know how to drive. In *Proceedings of the IEEE 2000 Intelligent Vehicles Symposium IV 2000*, pages 556–562, 2000.

[38] Y. Ruichek. Multilevel- and neural-network-based stereo-matching method for real-time obstacle detection using linear cameras. In *IEEE Transactions on Intelligent Transportation Systems*, volume 6, issue 1, pages 54–62. IEEE, March 2005.

[39] Y. Ruichek and J.-G. Postaire. Real-time neural vision for obstacle detection using linear cameras. In *Proceedings of the Intelligent Vehicles '95 Symposium*, pages 524–529, September 1995.

[40] D. Rumelhart, G. Hinton, and R. Williams. *Learning internal representations by error propagation.* MIT Press. Cambridge, MA, 1986.

[41] R. Sukthankar, J. Hancock, D.A. Pomerleau, and C. Thorpe. A simulation and design system for tactical driving algorithms. In *Proceedings of AI, Simulation and Planning in High Autonomy Systems*, 1996.

[42] R. Sukthankar, J. Hancock, and C. Thorpe. Tactical-level simulation for intelligent transportation systems. *Journal on Mathematical and Computer Modeling, Special Issue on ITS*, 27(9–11):228–242, 1998.

[43] R. Sukthankar, D.A. Pomerleau, and C. Thorpe. SHIVA: Simulated highways for intelligent vehicle algorithms. In *Proceedings of Intelligent Vehicles '95*, pages 332–337, September 1995.

[44] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K Lang. Phoneme recognition: Neural networks vs. hidden markov models. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, ICASSP-88*, volume 1, pages 107–110, 1988.

[45] P.J. Werbos. Backpropagation through time: what it does and how to do it. In *Proceedings of the IEEE*, volume 78, number 10, pages 1550–1560, 1990.

[46] P.J. Werbos. *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting (Adaptive and Learning Systems for Signal Processing, Communications and Control).* Wiley, 1994.

[47] J. Williams. A neural network controlled vehicle, that seeks a goal in an obstacle filled environment. Technical report, Department of Electrical & Electronic Engineering, University of Auckland, July 1992.

[48] Francis Wolinski, Frantz Vichot, and Mathieu Stricker. Using learning-based filters to detect rule-based filtering obsolescence.

[49] Shui-Ying Wong. TRAF-NETSIM: How it works and what it does. *ITE Journal*, 60(4):22–27, 1990.

[50] Cang Ye, N.H.C. Yung, and Danwei Wang. A fuzzy controller with supervised learning assisted reinforcement learning algorithm for obstacle avoidance. In *IEEE Transactions on Systems, Man and Cybernetics, Part B*, volume 33, issue 1, pages 17–27. IEEE, February 2003.

[51] Gening Yu and I.K. Sethi. Road-following with continuous learning. In *Proceedings of the Intelligent Vehicles '95 Symposium*, pages 412–417, 1995.