CHALMERS

UNIVERSITY OF GOTHENBURG

Driving Time Trial Laps using Neuroevolution

The development of a racing AI

Bachelor's thesis in Software Engineering

Gabriel Alpsten, Daniel Eineving, Martin Nilsson & Simon Petersson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2016

Bachelor of Science Thesis

Driving Time Trial Laps using Neuroevolution

The development of a racing AI

Gabriel Alpsten
Daniel Eineving
Martin Nilsson
Simon Petersson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
University of Gothenburg
Göteborg, Sweden 2016

Driving Time Trial Laps using Neuroevolution
The development of a racing AI
Gabriel Alpsten
Daniel Eineving
Martin Nilsson
Simon Petersson

Supervisor: K.V.S. Prasad, Department of Computer Science and Engineering
Examiner: Niklas Broberg, Department of Computer Science and Engineering

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg 2016

Driving Time Trial Laps using Neuroevolution
The development of a racing AI

Gabriel Alpsten
Daniel Eineving
Martin Nilsson
Simon Petersson
Department of Computer Science and Engineering,
Chalmers University of Technology
University of Gothenburg

Bachelor of Science Thesis

Abstract

Driving a race car competitively is a complex task. Programming a computer capable of solving this task optimally in every scenario is also difficult. Therefore it is interesting to investigate how well a machine learning algorithm is able to learn the most important behaviours from first principles. A simulator with simplified physics is utilised to train and assess the performance of the system.

An algorithm called Neuroevolution of Augmenting Topologies (NEAT) was used to train artificial neural networks. When the system steered a car which travelled at a constant speed, NEAT managed to find a reasonably effective behaviour that resembles professional racing tactics such as positioning and distance optimisation. However, when the system was used to both control the steering and the speed of the car, it drove cautiously and resembled professional tactics less. More efficient behaviours were found when the system was trained on shorter tracks. Additionally, a system that was trained on one track showed a considerable improvement in training times when migrated to a new track.

Some limitations of NEAT are discussed. The algorithm progresses gradually by a series of small improvements. It is observed that NEAT performs poorly when a composition of behaviours must be implemented simultaneously in order for the algorithm to progress. It is therefore advantageous if the problem is modelled to allow the algorithm to progress in gradual steps.

Keywords: NEAT, Neuroevolution, Reinforcement learning, Racing, Time trial.

Sammandrag

Att köra bil i ett racingsammanhang är en komplex uppgift. Att programmera en dator till att göra detta likaså. Det är därför intressant att undersöka hur maskininlärning kan användas för skapa ett datorsystem som kan lära sig själv att köra på en professionell nivå. En simulator med en förenklad fysikmotor användes för att träna och evaluera hur väl systemet presterar på uppgiften.

Algoritmen ”Neuroevolution of Augmenting Topologies” (NEAT), användes för att skapa och träna artificiella neurala nätverk. Dessa nätverk evaluerades med hjälp av simulatorn för att träna dem på att köra snabbt. När systemet styrde en bil som färdades med en konstant hastighet lyckades NEAT hitta ett tämligen effektivt beteende. Det uppnådda beteendet liknar hur en professionell förare skulle kört under samma förutsättningar. Beteenden som effektiv positionering inför kurvor och en minimering av körsträcka observerades. När systemet även fick tillgång till bilens gas och broms körde det försiktigare, och dess beteende var mindre likt en professionell förares. Effektivare beteenden hittades på kortare banor med endast en sväng. Utöver detta observerades det att tiden det tar för systemet att anpassa sig till en miljö minskas märkbart ifall systemet redan har tränat i en annan miljö.

Hur effektiv NEAT är i problemdomänen diskuteras. Algoritmen lär sig genom att gradvis förändra sitt beteende. Ifall förändringarna som krävs för att förbättra beteendet är för stora är det sannolikt att NEAT inte kommer lyckas hitta förbättringen. Därav är det fördelaktigt ifall problemet som NEAT appliceras på kan modelleras på så sätt att den optimala lösningen kan nås genom en serie av små förändringar.

Acknowledgements

Special thanks to our supervisor K. V. S. Prasad for his guidance during the project. We would also like to thank Niklas Broberg for taking time out of his busy schedule in order to provide helpful feedback. Additionaly we would like to thank Mikael Kågebäck for providing helpful feedback regarding machine learning.

Daniel, Gabriel, Martin, and Simon

Contents

Vocabulary

Actor An instance of an AI.
AI Artificial Intelligence.
ANN Artificial Neural Network.

FIA Fédération Internationale de l'Automobile, the international motorsports federation.

Genome A single neural network that is considered as an individual in evolutionary algorithms.

NEAT Neuroevolution through Augmenting Topologies.
Neuron Node in an Artificial Neural Network.

Overfitting Fitting a function to a special set of data excessively, making the function perform worse outside the given data set.

Time attack A sort of racing, were drivers compete against the clock. Only one car is on the track at any given time.
Topology The structure of a neural network.

Wavefront .OBJ 3D-mesh file format.

1
Introduction

We live in a dawning age of autonomous vehicles. During the last several years the development of autonomous cars has progressed immensely. Many car manufacturers are currently testing their autonomous car prototypes in real traffic and fully autonomous vehicles can be expected on the consumer market in the near future. At the heart of autonomous vehicles is the field of Artificial Intelligence (AI), especially in the form of machine learning; algorithms that learn.

The concept of machine learning is to create well-adapted AI systems that require the minimal amount of human design. The algorithms learn to predict correct answers within a problem domain by inferring knowledge from examples or from experience.

In autonomous cars, machine learning is used to solve tasks such as steering and image analysis [1, 2, 3]. In order to solve control tasks, such as steering and speed management, the system learns to emulate a specific behaviour by conforming itself to a large set of training examples. The training examples consists of pairs of system inputs, such as sensor data, and the corresponding correct control signals. These sets of training data can be gathered by driving the car while recording both the inputs from all the sensors and cameras on the car, and the control actions taken by the driver. Thus this training process requires the recording and quality assurance of vast amounts of example data.

There are other types of machine learning that learn without using example data. These algorithms learn by other means, such as learning from experience. One such type of machine learning is called reinforcement learning. Instead of sample data, reinforcement learning algorithms evaluate their behaviour by scoring it using a set of rules. Google used a reinforcement learning algorithm when they created the Go AI AlphaGo [4], which earlier this year won against the former world champion Lee Sedol [5]. This is an example of a reinforcement learning algorithm finding a behaviour that can be counter-intuitive or hard for a human to design.

One domain in which reinforcement learning could be used is racing. In competitive racing, drivers push the physical limits of their cars. The details matter, miniature differences in behaviour can accumulate to large time differences over the course of a lap. Machine learning could be used to find efficient driving behaviours. There is also the possibility of using machine learning to create the future generation of racing drivers. FIA, the governing body of Formula 1 and many other competitive racing series, has announced a racing series for autonomous cars [6]. The competing teams will use identical cars, except for the software which the teams can modify to gain an advantage. Thus the goal of the competition is to create the most effective autonomous racing system.

1.1 Problem

There are many potential applications of an autonomous racing system. The optimal racing behaviour depends on the properties of the car. A method that can find the optimal behaviour for a specific car would be useful for both the engineers and the drivers. Engineers could be provided with feedback on how modifications to the car affect the optimal behaviour. Drivers could learn from the optimal behaviour, for example by racing against this behaviour in a simulator. Thus giving the drivers the opportunity to adjust their driving to the specific car.

In theory the optimal racing behaviour is calculable. Given a specific car and track segment there exists an arbitrarily accurate function describing the time required to drive through the section. Given such a function, optimisation methods could be used to find the minimums in time spent. However, due to the complex nature of the problem, the number of variables in such a function is large, even if aspects such as the physical environment, e.g. temperature, oxygen levels, and wind, is neglected. The number of possible paths through a section is infinite so, in order to find an optimal behaviour in a reasonable amount of time the problem must be extensively simplified.

The complex nature of optimal racing presents quite a few other problems. Solving such complex problems manually is not easy, and implementing behaviour for every single scenario is not an option due to the large number of scenarios. Machine learning tries solves this problem by trying to find a general behaviour that adapts to the scenarios. Instead of manually implementing behaviour for every scenario, a general machine learning algorithm is implemented and presented with a general model of the problem and its environment.

Finding a general model for the problem and its environment can be very complicated. The information that is to be presented to the AI and the actions taken by the AI must be applicable in many different situations. The behaviour should not be optimal on a specific track, but rather effective in general. This requires the information that the AI is presented with to not let the AI learn track specific behaviour. One way to solve it is to let the AI make local decisions based on local information. This is similar to how humans drive a car; a person can see the shape of the road in front of the car and steer according to that. This is also true in racing, the driver sees the track and may also have a notion of how the track will proceed throughout the following corners.

1.1.1 Impact on Society

Solutions to these problems in the racing domain could also be applied in other areas of society. Technology advancements within racing may have a limited impact on society as a whole. However, improvements made on the race cars are sometimes later used in commercial vehicles. One such example is how steering wheels have become a control panel in road cars, by the influence of race cars [7]. Similarly, advancements in autonomous racing could be applied in commercial autonomous vehicles.

Every year more than a million people are killed in traffic accidents, furthermore

the economic cost of these accidents is several billion dollars [8]. Road transports and the transport sector in general are also major contributors to our societies impact on the environment [9]. Autonomous cars could potentially reduce these problems by preventing accidents and by being more efficient than human drivers [10].

1.2 Purpose

This project explores the possibilities of teaching an AI to perform time trial laps by operating a motor vehicle in a simulated environment. The goal is to evaluate whether machine learning can be used to effectively find general and optimal behaviours for time trial racing. Additionally, a simulator is developed in order to provide the AI with a flexible environment to operate in.

1.3 Limitation

The scope of this project is limited to developing an AI that finds the optimal behaviour of a race car during a time attack lap. Thus concepts in head to head competition such as racing strategy, overtaking, and pit stops are not considered. Additionally, the car is assumed to be in a pristine condition at all times. Thus aspects affecting the prolonged operation of the car such as fuel efficiency, tyre wear, or brake temperatures are not taken into consideration.

The problem of finding optimal behaviour can be solved by using many different types of machine learning. A combination of algorithms could potentially be used by breaking down the problem into parts. This project will not evaluate and compare different algorithms due to time limitations. Instead only one algorithm will be implemented and evaluated in depth.

The AI operates within a simulated environment. This allows for major improvements in time spent training and evaluating networks. The simulator is capable of evaluating networks significantly faster compared to controlling for example a radio-controlled car. By properly utilising the speed of modern computers, several networks may also be evaluated concurrently.

It is important that the simulator accurately simulates fundamental car behaviour. This is required in order to evaluate whether the behaviour found is reasonable. However, due to the complexity and time required to develop a simulator that accurately simulates every aspect of the real world, this project is limited to only implementing the fundamental aspects. Therefore, the simulator has been limited to include turning radius that depends on speed as well as acceleration and deceleration of the car. More complicated simulation aspects such as temperatures, oxygen levels and internal dynamics of the car, which have a limited effect on the fundamental behaviour have been left out.

1.4 Related Works

The Open Racing Simulator (TORCS) is an open source simulator that contains many racing tracks and implements graphics, AI drivers and more realistic physics than presented in this report [11]. Using TORCS may be an easy way to set up a working simulator and is probably useful in a project where more emphasis is put on realistic physics. Several academic studies discussing machine learning for racing have used TORCS as simulator.

D. Loiacono et al. (2010) present a machine learning competition for students, named "The 2009 simulated car racing championship". It contained a time trial lap and a racing final, using TORCS as simulator. The contributions provided a variety of control strategies. The most successful strategies had hard coded controllers built up by several single purpose components, or machine learning techniques, one of them NEAT. They also deal with aspects such as overtaking, gear control and spinning. The paper may therefore provide insight in how those problems may be addressed.

1.5 Bibliography Notes

This section presents comments some of sources that had a considerable on the project.

1.5.1 Racing

Edmondson (2011) and Beckman (1991) explain the underlying physics and theory of racing. These books explain how different laws of physics affect the properties of race cars, and how they affect the optimal driving behaviour.

1.5.2 Machine Learning

Haykin (1999) presents a solid foundation to neural networks and machine learning techniques in the book "Neural Networks: A comprehensive foundation". The book explains how neural networks work in detail. It also presents different machine learning paradigms and how they can be used in conjunction with neural networks.

Stavens (2011), Thrun et al (2006), and Huval et al (2015) explain how autonomous cars work and how machine learning is used to solve problems such as image analysis. These works explain how an autonomous car interprets its surroundings and make. Stavens (2011) and Thrun et al (2006) also explain how the award winning autonomous car Stanley works. This autonomous car system was developed at Stanford and was later used as the basis for Google's self-driving car project.

Stanley & Miikkulainen (2002) presents their algorithm NEAT. It is the first time the algorithm is presented, and the paper describes the mechanics of NEAT and benchmark it to other algorithms on the pole balancing problem.

Whiteson (2010) explains FS-NEAT, a modified version of NEAT that can find relevant input values significantly faster than NEAT. It states that it is beneficial

to start the NEAT process without an initial structure when the relevance of each input value is doubtful.

F. Gomez et al (2006) and F. Gomez et al (2008) present and discuss the algorithm Cooperative Synapse Neuroevolution (CoSyNE). The method train the weights in a fixed topology neural network. The weights are evolved in a cooperative fashion were a specific weight receive the fitness of the whole network it is located in. The author claim that it has better performance in dynamic control tasks involving large state spaces than non evolutionary reinforcement algorithms. CoSyNE also performed significantly better than NEAT on the pole balancing problem.

2
Theoretical Framework

The following sections will include an explanation of the underlying theory and physics of racing as well as an overview of and motivation for machine learning concepts used in the project.

2.1 Racing Theory

As mentioned in 1.2, the goal of the project is to create a racing AI capable of driving a car around a racing track as quickly as possible. Racing tracks are normally broad enough to allow many different paths for the driver. Different paths may differ in length but also affect the possible achievable speed. Time is the result of both distance and speed according to the following formula: $t = \frac{d}{v}$, where d is the distance driven and v is the average speed. The process of minimising the time is a process of both minimising the distance driven and maximising the speed.

The next subsection will briefly describe the physics of racing. It is a core part of understanding how professional drivers drive, as will be described in section 2.1.2. In section 2.1.3 the requirements on the simulator are discussed.

2.1.1 Underlying Physics of Racing

A moving car has momentum. In order to change the momentum, for example to accelerate, decelerate, or change the direction of movement, a force must be applied. These forces are applied through the tyres. This is a restricting factor for cars since the available traction, which is the amount of force that can be applied through the tyres, is limited. If the applied force exceeds the available traction, the tyres will slide or spin [12].

The brakes are generally very efficient and are mostly limited by the traction of the tyres, whereas accelerating is limited by the torque of the engine. Additionally, a car travelling forward is slowed by the air drag. When the car brakes, the tyres and the drag will work together, but an accelerating car has to work against the drag. These aspects make cars accelerate slower than they can decelerate.

As described by Newtonian mechanics, the kinetic energy of a moving object increases quadratically with the speed. A consequence of higher speed is that a greater force is required to change the momentum. The acceleration required to make a moving object stay in a circular motion is $a_c = \frac{v^2}{r}$ [12]. Using $F = ma$, the formula is transformed as shown in equation 2.1.

$$a_c = \frac{v^2}{r} \rightarrow \frac{F_c}{m} = \frac{v^2}{r} \rightarrow r = \frac{mv^2}{F_c} \tag{2.1}$$

Where a_c is the central acceleration, v is the speed, r is the radius of the circular motion, F_c is the central force and m is the mass of the car.

The car cannot turn as much when it accelerates or brakes. When a car accelerates or brakes, the weight of the car transfers slightly and the amount of pressure on the tyres will change, changing the traction capabilities. This limits the amount a car can accelerate or brake and turn at the same time [12]. The effect is further increased since both turning and changing the speed requires traction and share the same traction budget. Thus turning the car uses some of the available traction which reduces the amount the car can accelerate or brake [12, 13]. Due to the shape of a racing car, air drag contributes to the traction by pushing the car down. This effect, which is called down force, grows as the speed increases [12].

2.1.2 Introduction to Race Lines

A race line denotes the path a car drive around the track. It is an expression that include both the positioning and speed of the car. As mentioned in 2.1, driving fast and driving short are the key aspects to optimise in order to minimise lap times, but they sometimes counteract each other. If the car drives fast, the turning radius is increased and the car might need to drive a longer path, which may take longer time in total [13].

It is generally good to keep as high speed as possible and to drive close to the inner corners. The position where the race line is closest to the inner corner is called the apex. Depending on the situation the driver may take an early, middle, or late apex, as conceptually illustrated in figure 2.1.

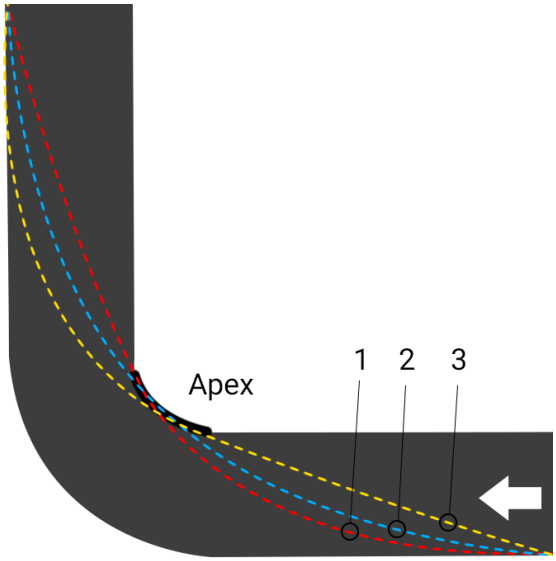


Figure 2.1: The point where the race line hit the inner corner is called the apex. Line 1 has a late apex point, line 2 a mid-apex point and line 3 an early apex point.

A driver usually needs to brake before a corner in order to keep the turning radius small and not crash into the corner. But, even at a suitable speed, the race line is seldom shaped as a circle sector. Competitive drivers do to some degree accelerate or brake at the same time as turning [13]. The traction budget is a limiting factor, but, if handled correctly, it can be used to increase speed and reduce distance driven.

If the car exits a corner with a higher speed, it can benefit from a reduced time spent in the next section. It is therefore generally beneficial to prioritise higher speed on longer sections [12, 13], as the difference in speed will accumulate to a larger time difference. The optimal behaviour is therefore to brake early so that a large portion of the turning can be done early in the corner, and then start to accelerate early, resulting in a late apex. This type of corner is often referred to as a type 1 corner [13].

However, when a corner ends a straight section and is not followed by another straight, it is normally best to brake and steer late to benefit more from the high speed before the corner [13]. This line will hit an early apex and it is often referred to as a type 2 corner.

The last type is the type 3 corner, which is any corner that do not fall into the type 1 or type 2 categories [13]. These corners are not adjacent to any straight section long enough for any of the two first race lines to be beneficial. The appearance of the optimal race line may vary drastically depending on the situation and the performance of the car.

2.1.3 Requirements on the Simulator

The purpose of this study is to evaluate how well a machine learning algorithm can learn the key aspects of an effective racing behaviour. Some of the aspects that will be evaluated are conceptual positioning throughout corners, timing and speed management.

If the behaviour of the AI is to be assessed in comparison to real world racing theory, the optimal behaviour in the simulated environment must be similar to the optimal behaviour in reality. The simulation may approximate certain aspects or neglect details, as long as the general characteristics remain, and the best practises in racing theory are still optimal.

Type 1 and 2 corners have typical race lines that conceptually do not depend on miniature differences in performance, although the exact positioning and timing do. In contrast, the optimal race line for type 3 corners vary largely depending on the situation and performance of the car. The behaviour in those corners might be interesting to analyse, but not in the purpose of comparing to the generic best practises.

Some fundamental aspects for the simulator is the turning radius and the relation between acceleration and deceleration. The turning radius forces the driver to control the speed appropriately and find a balance between the length of race lines and speed. It requires the driver to position the car well and to plan ahead. Acceleration need be slow enough for an increase of exit speed in type 1 corners to outperform a slightly shorter path, and also than accelerating, so that the speed prioritisation is correct for the type 1 and type 2 corners.

Weight transfer and the traction budget are important aspects to consider for a driver that drives to the limits of the cars performance. Not considering it in the simulation, would likely make the optimal timing for the different stages blend together slightly. It does, however, not change the general race lines for type 1 and type 2 corners as it only changes timing and positioning slightly. Aspects such as tyre wear and temperature changes the performance of the car and how much the driver can push the limits, but do not have a considerable effect on the local behaviour.

In summary, the key aspects to consider in the physics simulation is the turning radius and that the car accelerates slower than it brakes. Aspects such as weight transfer and the complexity the traction budget model would increase how well the simulator resembles reality, but would not change the generic behaviour for type 1 and type 2 corners.

2.2 Machine Learning

Machine learning is the field of study that concentrates on algorithms that can be said to learn [14]. This section will cover the machine learning theory and concepts that were considered and used within the project. Furthermore, the suitability of the different algorithms within the problem domain will be discussed.

2.2.1 Artificial Neural Networks as Knowledge Model

Machine learning algorithms require some representation of knowledge. One such knowledge model that is in wide use is the Artificial Neural Network (ANN). An ANN is a mathematical model that mimic the structure of the human brain [15].

The human brain is a large network of nerve cells called neurons. The neuron is a cell capable of firing an electrical pulse that can be transmitted to other neurons via connections called synapses. A neuron fires its pulse when the accumulated incoming signals from other neurons reach a certain threshold [15]. Akin to a biological neural network, an ANN is a network of artificial neurons or nodes. Each node has an output value which is calculated from a set of incoming connections. The connections are a set of weighted edges. The edges are directed, which means that they represent a signal flow from one neuron to another in the direction of the edge.

An ANN can represent a mathematical function by connecting a set of input nodes to a set of output nodes, thus representing a mapping from the input space to the output space. Between the input and output nodes, additional nodes called hidden nodes may exist. Output nodes and hidden nodes may not only be connected to the input nodes, but also other hidden nodes. These connections are what constitutes the neural networks.

The represented function is calculated by setting the values of the input nodes and then propagating the values through the network. The propagation of values works by feeding the value of one node to the others via the nodes outbound edges. The value of a node v is calculated by passing the weighted sum of the values on the incoming connections to an activation function $\phi(x)$. The full formula is shown in equation 2.2 where w_i is the weight and v_i is the value of an incoming connection.

Each neuron has a bias value b which is used as a base value in the activation function.

$$v = \phi(\sum_i [w_i v_i] + b) \tag{2.2}$$

The choice of activation function is arbitrary but will greatly affect the nature of the represented function. Usually a sigmoid-function is used, which is a smooth step function. A general sigmoid-function is described in equation 2.3. The value L denotes the lower bound of the function, a defines the scale or range of values and b defines the steepness of the function. The value-range of the function is $[L, L + a]$.

$$S(t) = L + \frac{a}{1 + e^{-bt}} \tag{2.3}$$

An ANN with at least two layers of hidden neurons and a sigmoid function as its activation function can be used to approximate any real function, furthermore the accuracy increases with the number of neurons [16]. The function approximated by an ANN can be changed by changing the topology or weights of the network. Artificial neural networks are thus useful for fitting curves to data.

2.2.2 Supervised Learning

Supervised learning is the process of learning with a teacher or learning from examples [15]. A large set of example data consisting of pairs of input configurations and the corresponding correct output is used. The learning process works by letting the knowledge model, for example a neural network, predict the correct output for given inputs in the data set. The knowledge model is then corrected in order to better predict the correct output. Algorithms that learn by induction often fall into the supervised learning category [14].

Supervised learning algorithms are useful when the goal is to create an accurate prediction model from a large set of example data. As mentioned in chapter 1, it is widely used in autonomous vehicles [1, 2, 3]. This is a good indication that it is possible to use for complex control tasks.

However, the primary goal of this project is to create an AI not by defining how it should behave but by defining what it should strive for. Supervised learning could certainly be used to create a racing AI that for example emulates a professional driver. It could also be used to solve one part of the problem, for example learning where to position the car on the track. However, learning from examples requires a set of examples that define the target behaviour, which is contradictory to the goals of this project. Furthermore, the scarcity of available data limits the use of these algorithms.

2.2.3 Unsupervised Learning

Unsupervised learning is a type of machine learning that in contrast to supervised learning does not learn to predict or approximate a correct output given an input, but rather learn to group or cluster sets of inputs into categories based on the input values [14]. These algorithms are prevalent in data mining and other areas where

clustering is useful. Clustering was not found to be useful in the racing domain, thus the paradigm was rejected in order to prioritise more suitable methods.

2.2.4 Reinforcement Learning

A central aspect of the learning process is evaluating the performance of the actor. Supervised learning algorithms compare the actors output with a set of correct values. However, if there is no data set available to train with the feedback must be acquired in some other way. Reinforcement learning algorithms solve this by scoring actors on how well they perform [17]. The set of example data used in supervised learning is replaced by some quantifiable measurement of performance.

There are many ways in which one can score actors on their performance. One problem that often occurs is that it is hard to provide actors with rewards based on local behaviour. Some actions are locally optimal, but globally sub-optimal. Thus in order to determine whether some action is beneficial it is often required to have knowledge about how this action affects the result globally.

In racing, rewarding local behaviour is especially difficult. Good behaviour is defined by how fast a lap can be completed. Thus a series of actions that may be beneficial for one corner, but results in an increase in lap time is not optimal. However, this is hard to determine locally. It would be favourable to utilise an algorithm that is capable of rewarding actors based on their global performance.

2.2.5 Markov Decision Problem

It is common to model reinforcement learning tasks as Markov Decision Problems (MDP). An MDP contain a set of discrete states which each have a set of transitions to other states. Two popular algorithms often used in this context are dynamic programming and Q-learning. Dynamic programming is used to cache intermediate results when doing a deep search through the state graph. Q-learning is based on training a neural network to estimate which transitions are most beneficial.

The racing problem is a dynamic control task and is best described as a continuous state and action space. Furthermore, it is a multidimensional problem with respect to the position and velocity vectors. Therefore, translating it to a discrete space, risk either to be an inaccurate approximation or require such a large number of states it is difficult to find good generalised behaviour [18].

There exist attempts to use MDP-based algorithms for continuous problems. One example is based on predicting states instead of defining them explicitly, as presented in the paper Practical Reinforcement Learning in Continuous Spaces [18]. This study shows promising results on the Mountain climbing car problem. However, the complexity of this problem is significantly lower compared to the racing domain, and it seems difficult to argue that the state prediction model presented is sufficient to handle the physics of the racing domain. But even so, if this or another way of modelling MDP works, the extra step of modelling seems uninteresting if there exists other, more direct approaches.

2.2.6 Neuroevolution

Neuroevolution is a set of reinforcement learning algorithms that learn by emulating evolutionary processes, for example natural selection. These algorithms learn by evolving neural networks to solve a specific task. The evolution process works by allowing advantageous traits to remain while disadvantageous traits are removed.

Determining whether or not a trait is advantageous is done by evaluating the performance of the AI. Performance is measured in terms of a fitness value. This value is calculated by a fitness function that is specific to the task at hand and should be defined by what defines good behaviour for that task. Furthermore, it should be defined such that increasing its value is equivalent to increasing the performance of the AI [19]. Defining the fitness function for a problem can be difficult. As described in 2.2.4 knowledge of the global behaviour is required in order to evaluate performance in racing. In some neuroevolution implementations such as CoSyNe and NEAT, this is solved by constructing a fitness function that evaluates performance on final results [20, 21]. By constructing a function that is based on the final results of evaluation, the AIs global performance can be evaluated in an easy way. For example, in racing the fitness function could be calculated with regards to the total distance driven as well as the average speed along this distance.

The evolution process can work on different levels of abstraction. For example, on the individual level, where well-adapted individuals are allowed to carry on their traits. It can also be done on the level of singular traits, for example connections in a neural network. In that case the traits that have a positive contribution to the performance of the AI are kept while negative traits are removed. This type of machine learning has been shown to be effective in comparison to other machine learning algorithms at solving non-linear control tasks such as the pole balancing problem [22].

Neuroevolution seems like a suitable machine learning paradigm for the racing domain based on three properties. Firstly, the possibility of evaluating actors on a global level. Secondly, the possibility of using a continuous model of the problem. Lastly, neuroevolution algorithms have been shown to be effective at solving non-linear control problems [22].

2.2.7 Neuroevolution of Augmenting Topologies

One interesting neuroevolution algorithm is Neuroevolution of Augmenting Topologies (NEAT). The algorithm was developed by K.O. Stanley and Risto Miikkulainen at the University of Texas [21]. One interesting aspect of NEAT is that the algorithm constructs network topologies from a minimal structure. This removes the need to design or choose a network topology. It also results in smaller neural networks, which can make it easier to reason about how the networks work. NEAT is also easy to adapt to different problem domains since it is loosely coupled to the problem it solves. The only interaction between the algorithm and the problem domain is the evaluation process where the networks are scored. Because of these properties, NEAT was chosen as the machine learning algorithm for this project.

In NEAT the evolution process is simulated on a pool of genomes. A genome is the genetic encoding or DNA of an ANN consisting of a number of genes. Each gene

represents a connection in a neural network. The genomes are grouped into species based on genetic similarity. A genome only competes with the other members of its species. The introduction of species protects the diversity of the population, which has been showed to improve the learning rate of the algorithm [21].

The evolution process works by culling each species based on the evaluated fitness of each genome. The fitness of each genome is evaluated by a detached and separately implemented algorithm. The worse performing half of each species is removed, the better half is allowed to pass on their genes and the best genome is kept as it is. Some species are removed due to stagnation or due to performing significantly worse than the average. The other species are allowed to breed a number of children. The number of children a species is allowed to breed is calculated from the relative average fitness of the species.

The breeding process used in NEAT creates children by combining genes from two parent genomes and then mutating the child. The mutation is a stochastic process which may change a genome in several ways. The possible mutations are the addition of a new gene, adding a new node by splitting a gene into two, modifying the weight of a connection, disabling an enabled gene, or enabling a disabled one. The iterative process of evaluating, culling, breeding, and mutating, generates a pool of genomes. Each new pool generated is referred to as a generation.

Another feature of NEAT is that the neural networks are constructed from a minimal initial structure. By gradually augmenting the topology of the network the resulting size of the network can remain small. Additionally, only the beneficial modifications will survive the evolutionary process. Thus useless features will be discarded, further reducing the size of the networks. A minimal structure reduces the search space of the algorithm, since there are less connections and weights to optimise. Additionally, a small network structure allows for a deeper understanding for how the network operates by examining the network representation. This property is beneficial since it simplifies the analysis of the neural networks.

3
Implementation & Experiments

In order to evaluate how NEAT can be used to find racing behaviours a number of experiments were conducted. The experiments require a controlled environment which can be reasoned about, and where results can be reproduced. This chapter will cover the implementation of an experiment suite, which includes a racing simulator, an implementation of NEAT, a visualisation system and a control layer which connects the system components. Furthermore, the reasoning behind and execution of the experiments that were conducted will be presented.

3.1 Implementation of the Simulator

The simulation is an iterative process where the state of the universe, which only consists of the car and the track, is updated continually. Each update represents a time interval of 10 milliseconds within the simulated universe. The updates are carried out in direct sequence, thus the simulation rate depends on the clock frequency of the computer running the simulation. However, with the speed of today's computers, multiple updates can be carried out every second. This enables a desktop computer to simulate years of racing in a few hours.

The simulated car is represented by a simplified model of a real life car. Only the required properties discussed in section 2.1.3 are simulated. Each update of the car state results in an update of the cars position according to equation 3.1

$$\vec{p}_{n+1} = \vec{p}_n + \vec{v}\Delta t \tag{3.1}$$

Where $\vec{p}_i$ is the position vector of the car in iteration i, $\vec{v}$ is the velocity vector of the car and Δt is the delta time or the time-step, which is fixed at 10 milliseconds. This frequency yields a sufficiently high time resolution. At a speed of 100 m/s, which is slightly higher than the top speed used in the simulator, the car would travel one metre per iteration. In relation to the size of the car and the track, one metre is a short distance.

The velocity vector $\vec{v}$ is updated due to acceleration based on the forces acting on the car. According to classical mechanics, the acceleration a of an object is equal to the applied force F, divided by the mass m of the object, a relationship which is described by the equation $F = ma$. This can be extended into higher dimensions by representing the applied force and the acceleration as vectors with one value per axis. Thus the acceleration vector can be found with equation 3.2.

$$\vec{a} = \frac{\vec{F}}{m} \tag{3.2}$$

The car weight used in the simulation is 642 kilogrammes, which albeit being oddly specific, is a reasonable weight for a Formula 1 car without a driver. The rules state that the minimum weight of a car including the driver in full gear but excluding fuel is 702 kilogrammes [23]. The accelerating force was set to a constant 9100 newton, the braking force 25000 newton and the possible centripetal force 2500 newton.

How much the car is able to turn, or rotate, depends on the current turning radius. The angle the car rotates depends on how far it drives along the circle sector, approximated by $v\Delta t$. As the length of a circle sector is θr, were θ is the angle and r the radius of the circle, $\theta = \frac{vt}{r}$. Combined with equation 2.1 the resulting function for the rotation is described in equation 3.3. Skidding of the car is excluded from the simulation as it only represents a state where the traction capabilities have been breached.

$$\Delta r = \frac{F_c \Delta t}{mv} \tag{3.3}$$

The tracks used in the simulator are flat 3D-meshes with triangle faces. The meshes were modelled in Autodesk Maya and were imported to the simulator in the Wavefront .OBJ format. In order to simplify the representation of the track the cross sections of the track are extracted from the mesh. These cross sections are treated as a series of checkpoints that define the track.

3.2 NEAT Implementation

NEAT was implemented in C++ primarily based on the descriptions in the original paper [21]. Two implementations were also used as references, the latest C++ implementation from the authors themselves [24] and a script written in Lua used in a Super Mario bot [25].

In order to verify the implementation of the algorithm it was tested by training it to approximate the logical exclusive-or function (XOR). The reason that the XOR-function is relevant to test is because it is not linearly separable, this means that a neural network requires hidden nodes in order to approximate the function [15, 21]. The test was used to validate that the implementation of the algorithm made additions and changes to the network structure.

Worth noting is that the neural networks used by NEAT differs slightly from the classical approach. Instead of treating the biases as a separate set of values, a bias input is introduced. The bias input is set to a constant non-zero value. In order to add a bias value to a neuron a connection from the bias input to that node can be added. The bias value of the node can be changed by modifying the connection weight. This is a convenient approach since it allows the algorithm to modify the bias values with the same operations that are used to modify the network topology.

Additionally, some minor utility features were added to our implementation in order to allow for a more flexible training process. For example, the activation function used in the neural networks is specified as a lambda-function, thus it can

be changed at run-time. Therefore, specific activation functions could be used for a specific experiment. Furthermore, the ability to toggle the creation of an initial structure for new networks was added. If the option is enabled, the genomes start with a lattice of connections that connect each input to each output with a randomised weight. The variant of the algorithm where the initial structure is omitted is called FS-NEAT [17].

3.2.1 NEAT Configuration

NEAT can be configured in order to change the general behaviour of the algorithm. The configuration used during the project can be found in appendix A.1. This configuration is based on the Super Mario bot implementation [25] as well as the original paper [21]. However, some modifications were performed in order to achieve a more appropriate behaviour.

It was noted that the amount of nodes produced by NEAT was significantly larger than required. Nodes were produced when a simple connection weight modification would have been more appropriate. To prevent this behaviour, the probability of adding a node was decreased.

When performing long training sessions, it also seemed reasonable that allowing networks to stay alive for longer periods of time would be beneficial. This could allow behaviours that initially are non-optimal to prove themselves superior when allowed more time to evolve. Thus the allowed time for a network to live on, before it is killed off by stagnation, was increased.

3.3 Training Process

The training process adapts neural networks to drive a car. NEAT is utilised in combination with the simulator to train the genomes. The actual learning, i.e. changes to the genomes, is performed by NEAT. However, the evolutionary process requires a measurement of fitness to evaluate genomes. Thus the training procedure is responsible for calculating the fitness of every genome, so that the evolutionary process can proceed.

In order for the training process to evaluate a genome and calculate its fitness, the simulator is utilised. Calculating the fitness requires information about how well a specific genome has performed. The training process generates a network based on the genome that is about to be evaluated, and provides it to the simulator. The simulator uses this network to control the car during the simulation. In each update of the simulation, the simulator feeds data about the car and its environment into the network, with which the network calculates how to control the car. Once the simulation terminates, the simulator presents the training process with a result that can be used to calculate the fitness.

The following subsections cover the network inputs and outputs used in detail. Furthermore, the evaluation process and fitness function will be covered.

3.3.1 Interpretation

The system requires information about the car and environment in order to decide how to act. Providing the appropriate information, is essential if the system is to learn an effective behaviour. The appropriateness of information depends on the problem and target behaviour.

The inputs used within the project aim to give an easy way for the system to couple situations with an appropriate action. The input values are summarised in table 3.1 and can be divided into two groups; one that describes the shape of the track in front of the car, and one that describes the current state of the car. Even though not all inputs were used in every experiment, they are all present in at least one experiment each.

Name	Description
Track curvature	10 angles describing the curvature of the track.
Sum of curvature	Three sums of track curvature, angles 1-4, 4-7 and 7-10
Car speed	The current speed of the car
Angle to mid line	Angle between the car direction and the track mid line
Dist. to middle	Distance between the car and the track mid line
Dist. to left edge	Distance between the car and the left edge of the track
Dist. to right edge	Distance between the car and the right edge of the track
Bias	The value 1 to be used as a constant base value in neurons

Table 3.1: List of all available inputs for the neural network.

There are two types of input values describing the shape of the track in front of the car. These inputs aim towards letting the system plan ahead for the car, in order to adjust position of the car before arriving at corners.

The first type is the curvature data, which is a series of angles. The angles are based on the position of the car and 10 points on the mid line of the track. The first angle is calculated from the direction that the car faces and the direction to the first point. The rest of the angles are calculated by comparing the directions of the succeeding line segments defined by the previous points.

The distance between the points increase exponentially for each point. This leads to higher resolution closer to the car where detail is more important, and lower resolution where the general outline of the track is sufficient. For the experiments conducted, an increase of 35% was deemed sufficient with points spanning from 10 meters to 546 meters in front of the car, with the spread presented in table 3.2. This distribution of points would give the car the chance to interpret the curvature even further than necessary to plan ahead.

Point	Distance [m]
1	10
2	24
3	42
4	66
5	100
6	144
7	205
8	287
9	397
10	546

Table 3.2: Distance from the car to each point as given input.

The second type is the summed curvature of three different regions. The regions

consist of the points 1 through 4, 4 through 7, and 7 through 10. These summations provide a better overview of the track shape than the individual angles.

The set of inputs that describe the current status of the car are more simple in nature. They aim towards giving the system the capability of keeping the car on the track at all times. These inputs consist of current car speed measured in meters per second, angle between the cars current direction and the mid line of the track, distance from the car to the tracks mid line, as well as the distance between the car and the left and right edge of the track.

Additionally, a bias value is introduced as an input to the network. The value of this input will always be 1. This is due to the fact that if all inputs to a specific neuron are zero, the neuron would not be able to produce a signal. Since the output of a neuron is the weighted sum of all its inputs, the bias can be used to produce a base value for the sum.

Name	Description
Turning rate	Current steering applied to the car
Acceleration & braking	Amount of acceleration/braking applied to the car

Table 3.3: List of all available outputs for the neural network.

The number output values for the networks used during the project are limited to two. One for turning rate and one for acceleration and braking. An activation function was chosen such that both output values can span from $[-1, 1]$. Specifically, the activation function seen in equation 3.4 was chosen.

$$S(t) = \frac{2}{1 + e^{-t}} - 1 \tag{3.4}$$

For turning rate negative values represents steering right and positive ones to the left. If the output value of a network exceeds the amount of steering that can be applied at the current car speed, the simulator floors the value to that maximum.

For acceleration and braking, a negative value represents braking, while a positive value represents accelerating. The value is interpreted by the simulator as the ratio between the desired and the maximum value, thus a value of 1.0 correlates to the maximum amount of acceleration.

3.3.2 Evaluation

When calculating the fitness, two values from the simulation are considered; the distance progressed on the track and the time spent before termination. Since the evolution of the genomes happen gradually, the performance must also be allowed to progress gradually. Rewarding for distance driven gives an incentive to complete the track and not crash. Time is needed as it distinguishes cars that reach the same distance. After some level of basic improvements in time, advanced racing behaviours will be required to improve further.

However, if time is valued in combination with progression it might incite to drive fast and crash instead of progressing in distance. One way to solve this is

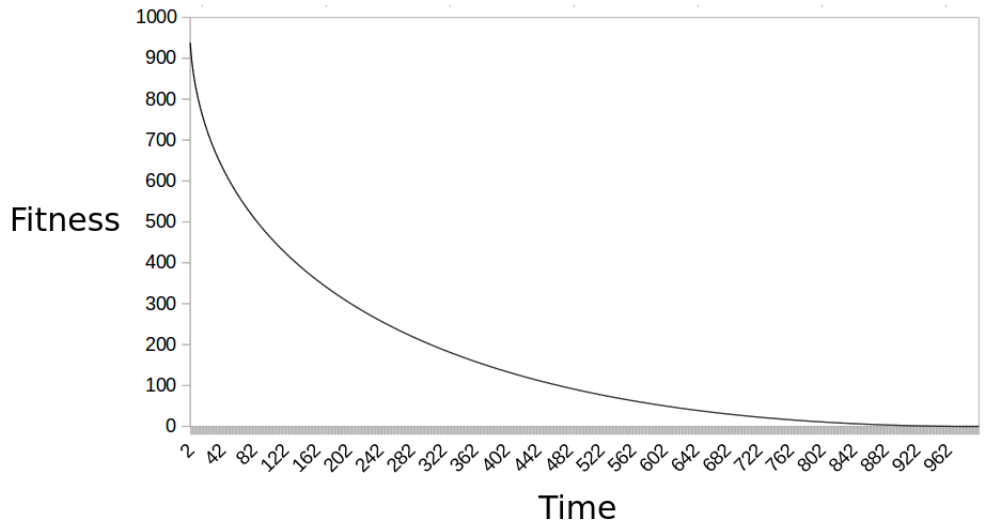


Figure 3.1: Example use of the time contribution in the fitness function with $t_{max} = 1000$. The fitness decreases with time, as well as the resolution of the function.

to only consider time it manages to finish the track. In that way, fitness is first rewarded for the progression on the track and then, if the car also completed the track, extra fitness is added for how little time was spent.

As of this, the time contribution to the fitness function is always positive. In addition, as time get smaller it will get increasingly difficult to decrease it further. It may therefore be beneficial if the resolution of the fitness function is greater for smaller time values. One function that satisfies these conditions is $(\sqrt{t_{max}} - \sqrt{t})^2$ presented in figure 3.1, where t is the termination time of the simulation and t_{max} is a suitable upper limit for the track. Combined with distance, the resulting fitness function is as described in equation 3.5.

$$f(d,t) = \begin{cases} d & \text{if } d < d_{max} \text{ or } t \geq t_{max} \\ d_{max} + \lambda * (\sqrt{t_{max}} - \sqrt{t})^2 & \text{else} \end{cases} \tag{3.5}$$

Where the distance driven along the track is denoted by d and the distance to complete a lap is denoted by d_{max}. In order to properly balance the importance of lap time and distance driven, the constant λ was introduced. During the following experiments, $\lambda = 5$ was found to be reasonable constant.

It is important to note that the distance driven d is the distance that the car has driven along the track. This distance is equal to the position of the car, projected on the mid line of the track. Thus driving in a sinusoidal pattern will not affect the final distance driven.

3.4 Experiments

The project was an iterative process. Progress was made by experimentation, where different ways to model the problem and train the AI were tried out. In order

to establish which aspects of the target behaviour that can be found with NEAT, a number of experiments of varying complexity were conducted. The following subsections will motivate the need for the different experiments and explain how they were conducted.

3.4.1 Steering a Car Moving at a Constant Speed

Driving a car is a complex problem, consisting of several sub-problems. This experiment aims to reduce the problem to one of its components, the steering. Instead of learning to control every aspect of the car, the AI will learn to steer a car that travels forward at a constant speed. The speed is sufficiently low to allow the car to steer through every corner on the circuit.

The track can be seen in figure 3.2. The corners are designed to have an increasing complexity along the track, and consists of a variety of corners. The input points shown in table 3.2 on page 20, covers approximately the two closest upcoming corners, regardless of what section the car is located in.

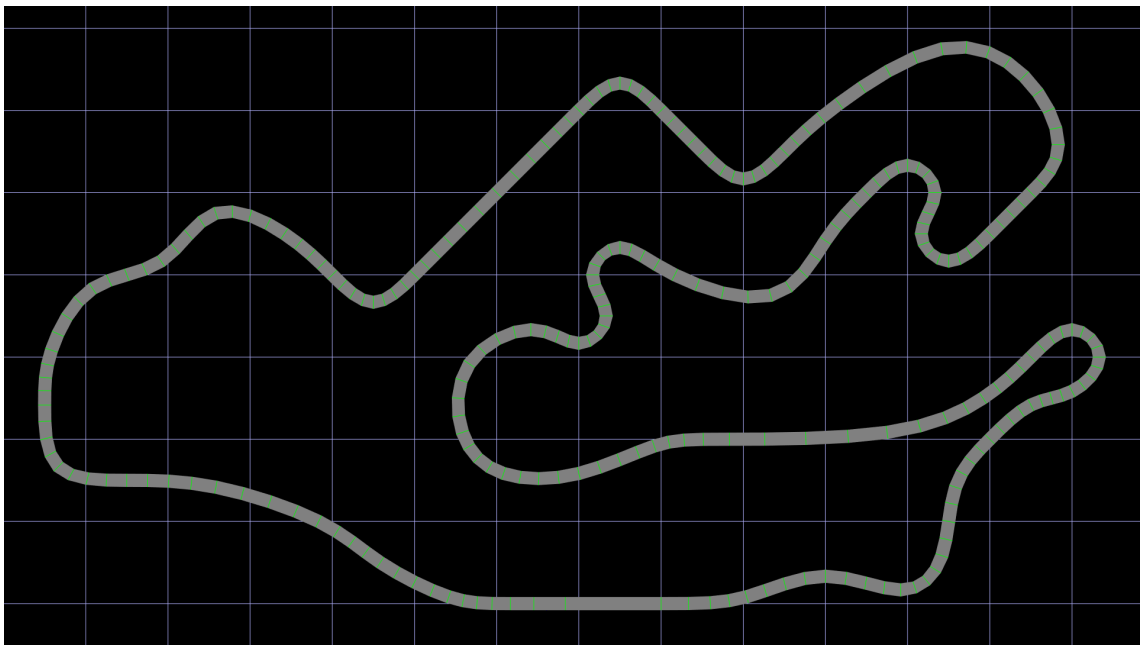


Figure 3.2: Circuit used in constant speed and full control experiment

Two variations of this experiment were conducted at several car speeds. In the first variant, the system learns to steer but is only given local perception of the environment, which in this case means that the car knows its position and orientation relative to the road. In the second variant the number of inputs is increased and the car is given the curve data. Thus the car receives information about the shape of the track ahead.

These experiments will give insight into how well the AI can interpret the provided data and how an increased perception of the track shape affect the behaviour. If the AI successfully learns to steer the car it would ensure that such behaviours can be found with NEAT. Furthermore, it would show which of the provided inputs

that are relevant to solving the problem. If the AI learns to steer only using a subset of the inputs, it shows that the other inputs are superfluous.

Comparing the result of the different experiment variants should show whether or not the addition of curve data will enable the AI to plan ahead and how that affects the behaviour of the AI. Without any curvature information, the AI is unable to plan ahead. Since it is only aware of the current position and orientation of the car, it is only able to react to that information. It seems reasonable that providing information describing the shape of the track ahead of the car will give the car the chance to plan ahead. Thus improving the AI's ability to take difficult corners and steer effectively.

3.4.2 Full Control of the Car

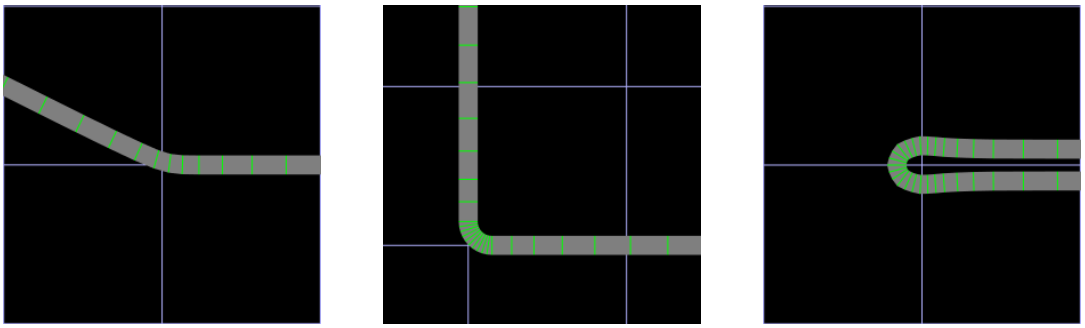
In this experiment, the AI will train on driving with full control of the car. In addition to steering it will also control the speed by accelerating and braking. In addition to the position, orientation, and curvature data the AI will be provided with the current speed of the car. The result of the experiment will further establish the limits of the training system.

Several experiment variations with this configuration will be carried out. In the first variant, which is the baseline experiment, the AI will drive on the same complete circuit as used in the constant speed experiments. This will show whether or not the AI is able to stay on the track to the same extent as when it only steers. Furthermore, it will show whether or not the system can learn to manage the speed in an effective way. If the system is able to find an effective behaviour, the natural response would be to even further increase the complexity of the problem. However, if no effective behaviour is found, it indicates that the training process and system configuration are suboptimal.

3.4.3 Short Track Segments

This variant of the experiment will test the system on shorter track segments, only containing one corner each. The motivation for testing short tracks is that it should be easier for the system to find effective behaviours on a short segment than on a complete circuit. This is because the behaviour required to complete a long circuit is generally more complex than the behaviour needed to complete just one corner. Thus, the algorithm will be able to start optimising for speed earlier and on less developed genomes.

The tracks consist of two straights connected by a corner, and can be seen in figure 3.3. The straights are between 400 and 500 metres long and there are three types of corners used. The first corner is a 30-degree corner throughout which the car should be able to keep a high speed. The second one is a 90-degree corner with a small inner radius. The third corner used is a hairpin corner which turns 180-degrees with a small inner radius.



(a) 30-degree corner.

(b) Tight 90-degree corner.

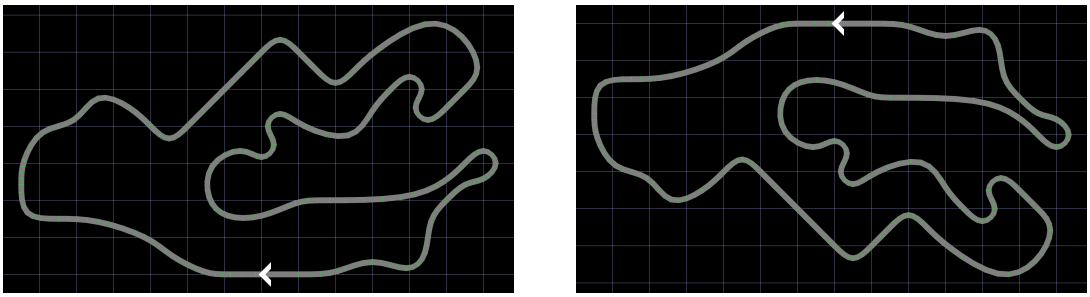
(c) 180-degree hairpin corner.

Figure 3.3: Short track segments with different corner types. The grey line is the track.

Successful results in these experiments would confirm that the system can find effective behaviours for shorter track segments. If that is the case the training process for the short track segments could be extended to solve more complex problems. A scenario where the experiment yield significantly more effective behaviours than the baseline experiment would give insight into the limits of the training process, and how the fitness function is used. This insight could be used to find effective behaviours for longer tracks.

3.4.4 Mirrored Track

One of the project goals is to find generalised behaviour of how to drive in a racing domain. This means that a genome learns to drive regardless of what track it drives on. Instead of finding behaviours effective on a specific circuit, the goal is to find a behaviour that is effective on any circuit. In order to test this a population of genomes is trained on the complete circuit used in previous experiments. Eventually the population will have adapted to the circuit. The population is then transferred to a new circuit, which as shown in figure 3.4 is a mirrored version of the one used earlier. This means that the genomes will drive on a track where each corner on the circuit has the same shape but opposite direction of the ones on the other circuit. The development of this population can then be compared to the development of a control group that is only trained on the mirrored track. Note that the number of tests and variation in environment is not enough to prove generality. However, the results will indicate whether or not some degree of generality is obtainable.



(a) The original circuit. **(b)** The mirrored circuit.

Figure 3.4: The original and mirrored versions of the circuit. The arrows show the starting position and direction of the car on each track.

There is a wide range of possible results in this experiment, in the worst case, the performance of the trained group is equal or worse than the control group and in the best case it performs as well as before the change of circuit. The best case scenario is unlikely, since the genomes will encounter corners they have never encountered before. However, if the test population outperforms the control group in any significant way it shows that the population has acquired some general knowledge of how to turn or manage the speed.

If the results show some form of generalisation there is a possible extension to the experiment that could yield interesting results. The extension regards the extent to which the population forgets knowledge about the original circuit when migrated. If the population is migrated back to the original track after learning how to drive on the mirrored one, will it still remember how to drive on it? Can the population learn to drive well on one track without forgetting how to drive well on the other?

4
Results & Discussion

The experiments performed during the project have led to many interesting results. Each experiment has contributed to improving the system. Either by highlighting issues in the training process or by providing a way to evaluate modelling decisions.

This chapter presents and discusses the results achieved during the project. The chapter is divided into sections by experiments performed. Each section presents and discusses the results of an experiment. The sections also introduce how the results relate to each other. The order in which these results are presented follows the order in which conclusions were drawn and the project progressed.

4.1 Steering a Car Moving at a Constant Speed

As explained in 3.4.1, two experiments where the car moved at a constant speed were conducted during the project. The following sections presents the results achieved during these experiments. Section 4.1.1 presents the behaviour of a system that is not presented with information about the curvature, while section 4.1.2 presents the behaviour of a system that is, as well as a comparison between the two results. Additionally, an observation made when removing the limitations of the cars turning radius is presented in section 4.1.3.

4.1.1 Only Local Perception

Providing the system only with local perception and giving it control of the steering, as explained in 3.4.1, present interesting results. The system immediately learns that positioning the car in the middle of the track at all times is preferable. Given that the system only has knowledge about the local relation between the car and the track, this behaviour is reasonable.

On straights, this behaviour works well, it could even be considered effective in some scenarios. However, in corners it is not as effective. When the track turns the car will drift away from the middle line. The system responds by steering back toward the middle. Before the system have had sufficient training, the amount of steering is either to large or too small. If the AI steers too much, the car will cross the middle line, and then have to react to that by steering the other way. This leads to the car moving in a sinusoidal pattern along the track, which eventually causes the car to crash. To small of an adjustment instead results in the car not getting back to the middle, which also eventually leads to a crash. It has been observed that moving in a sinusoidal pattern occasionally helps the car to complete a complicated

corner, by positioning the car in a good way. Even though this behaviour improves fitness, and in theory could make the car complete the lap with a higher constant speed, it is not an effective or optimal behaviour, thus it is not the target behaviour.

The amount of training required to complete a lap, if possible, is dependent on the constant speed of the car as seen in table 4.1. When the car manages to complete a lap, the sinusoidal patterns stops due to the amount of steering applied being finely tuned to stay as close to the middle as possible. For speeds higher than 11.7 m/s, the system never manages to complete a lap. This is due to the cars turning radius being too large to complete every corner by staying to the tracks middle line. Some degree of sinusoidal patterns is beneficial in these scenarios, since they occasionally cause the allowed turning radius through a corner to be larger.

Speed [m/s]	Generations
10.0	2
11.6	5
11.7	22

Table 4.1: Data of how many generations required until a genome with a constant speed could take a complete lap with local perception.

These results confirm that NEAT can find behaviours that are applicable to steering a race car. The system managed the task better than expected. However, due to the limited information, the system is unable to prepare for an approaching corner in a sophisticated way. The only information the AI can base its decisions on is its current position on the track. Providing information about how the track will progress should enable the system to prepare for upcoming corners.

4.1.2 Local Perception and Track Curvature

When using the track curvature data in addition to the local perception, solutions that completed the track was found up to 12.1 m/s. It is 0.4 m/s more than was found not using the curvature data. The difference in speed may seem small, but the difference in behaviour is substantial.

What the solutions found generally do is that they position themselves well before a curve, to compensate for the larger turning radius required. They also start to steer before the curve actually start.

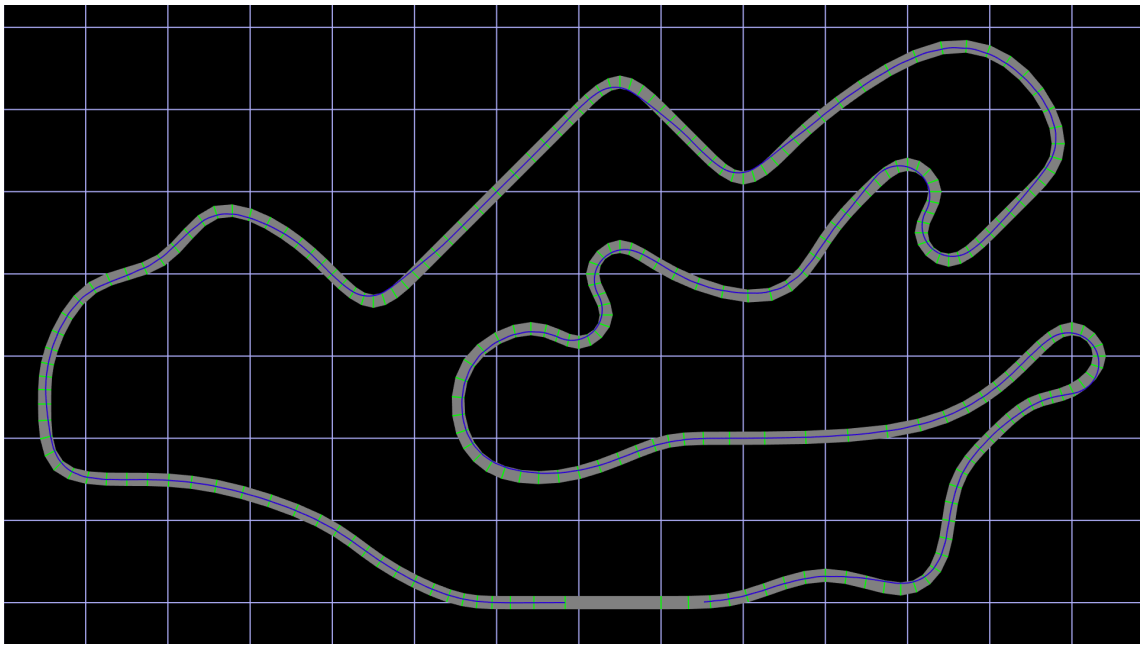


Figure 4.1: Race line showing a clockwise lap performed by the best genome found at the constant speed 12.0 m/s.

At 12.0 m/s, the system finds an effective behaviour where the AI steers early in the corners as showed in figure 4.1. At this speed the AI has to push the limits in order to steer through the sharpest corners, as showed in figure 4.2. The system managed to complete the whole circuit after 356 generations. The fastest lap time achieved was 415 seconds, which was reached after 1225 generations. Compare this to the 433 seconds it would take for the car if it was able to travel along the middle line of the circuit at the same speed. The behaviour found at this speed can be considered as close to optimal. The AI behaves effectively in corners. However, it is possible to drive shorter paths through some sections.

Generation	Time [s]
356	417.49
1000	415.70
1225	415.02

Table 4.2: Showing the best lap times with constant speed 12.0 m/s, at different stages of the training process.

At higher speeds, the turning radius became too large to steer through the sharper corners effectively. The AI had to turn very late in the section displayed in figure 4.2 in order to make it through the second corner. This led to an ineffective behaviour since the AI took late turns in the other corners as well. It also made it less interesting to investigate further, since no effective behaviour was found.

Each of the solutions found showed some typical characteristics. If one solution managed to drive close to the inner side in a sharp corner or positioned itself remarkably before a corner, it also showed that tendency for all other major corners.

Similarly, if the AI took late turns in the sharp corners, it took late turns in all corners.

The recurrent characteristics in the behaviour for a particular solution often had one limitation, that they failed to manage simpler corners as efficient as the tougher ones. Even though near optimal behaviours were observed in the tougher corners, no similarly optimal behaviour was observed in the easier corners. In the simpler sections the system stays close to the middle of the track instead of taking the shorter path along the inner edge.

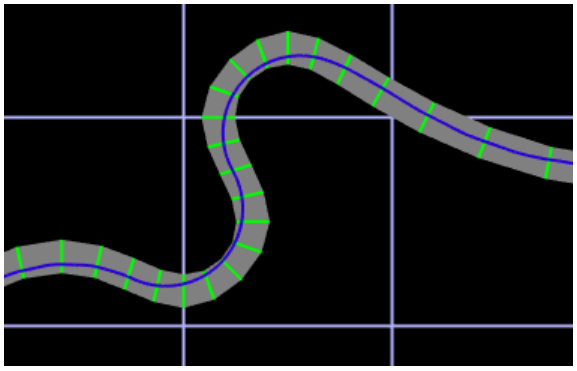


Figure 4.2: Race line showing how the best genome found at a constant speed of 12.0 m/s, drove through the section. The section is approached from the right.

It seems like the system does not properly learn to distinguish between different levels of corner difficulty. Instead the system finds one behaviour and scales it depending on the situation. Worth noting is that these training sessions only lasted for up to 600 generations. It is possible that more effective behaviours could be found during longer training sessions.

4.1.3 Shortest Path

The fastest route for a car that moves forward with a constant speed is the shortest one. This means that the only way for a genome to increase its fitness once it completes the circuit is to decrease the distance driven around the track.

The result show that after the algorithm finds specimens that are able to complete the circuit, the gene pool continues to improve. The path around the circuit is shortened to a great extent. The difference in length of the race line and the middle line of the circuit is significant.

The optimal behaviour is intuitively to always drive on the inner curves and to drive a straight line between the curves where it has to turn. Small curvature variations should not matter unless the car is required to steer in order to stay on track.

The car follows the key behaviour aspects, but not to the extent that the path is optimal. We can see that it drives tightly to the inner side for very tight curves, but not for low intensity curves.

4.2 Full Control of the Car

The base experiment performed, as explained in section 3.4.2, gives the possibility of controlling speed by controlling the throttle and brakes of the car. The addition of the speed management led to a larger increase in complexity than anticipated. By looking at the results from training sessions, a significant increase in complexity is apparent.

It takes 98 generations until the training process reaches a point where one genome can perform a complete lap. The behaviour observed after 98 generations is one that drives at a constant, slow pace along the track. The car has a fitness value of 5346.8, which means it travels 5200 meters in 1110 seconds, as seen in table 4.3. Between the corners, the car travels in a straight line along the track. The position at which this line is relative to the tracks mid line changes when the car goes through a corner.

The data shows that the system improves after it manages to complete a lap. The system finds genomes able to drive faster and faster around the circuit. After the system learns to complete the circuit, the rate of improvement is slowed down. As seen in figure 4.3, the best genome of each generation increases rapidly until the fitness exceeds fitness of 5200. When this fitness level is reached, the rate of improvement is flattened to a more gradual curve.

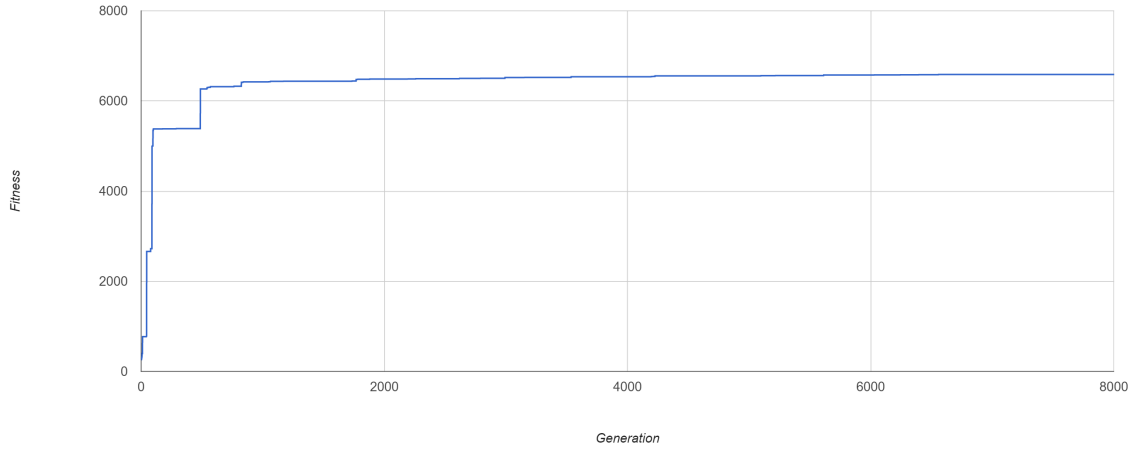


Figure 4.3: Showing the best fitness of each generation. Note that after 98 generations the fitness value surpasses 5200 as a genome has completed the circuit.

Generation	Fitness	Time [s]
98	5346.81	1110
1000	6423.52	533
8331	6587.56	491

Table 4.3: Significant results from full control experiment.

A fitness of 6587.56 was the highest value that was reached during this experiment. Reaching this fitness value took 8331 generations, and it corresponds to completing the track in 491 seconds. Table 4.3 shows that the system manages to lower the lap time significantly. However, the final result as seen in figure 4.4, is not optimal. Observations from the training process also shows that the path taken by the car at generation 8331, still resembles the one that was found at generation 98.

The decrease in lap time has been observed to be based on two improvements. The first one being the fact that the average speed around the track has increased significantly, and the second one being the increase in acceleration and braking

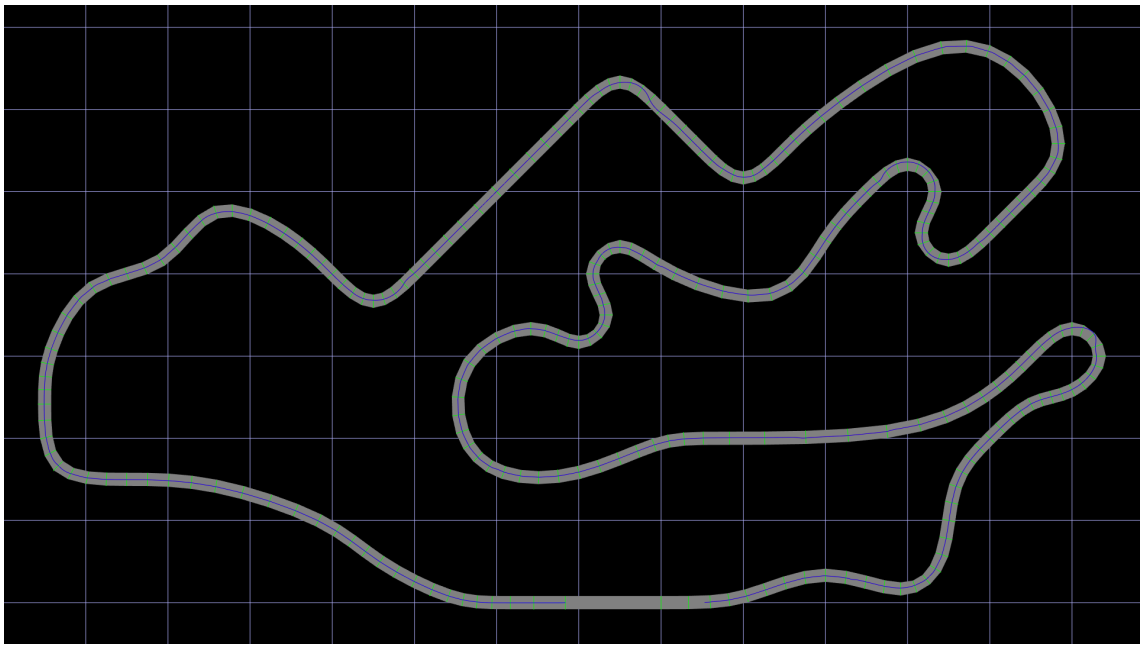


Figure 4.4: Showing a clockwise race line completed by a genome with a fitness value of 6587.56, the best fitness reached in the steer and speed control experiment. The speed of the car is presented by the line colour, with blue as slowest possible speed and red as max speed.

during turns. It has been observed that the car accelerates during the straights and brakes before corners, although very small amounts of acceleration and braking. In order for this behaviour to properly increase the performance of the network, it would have to be more distinguishable.

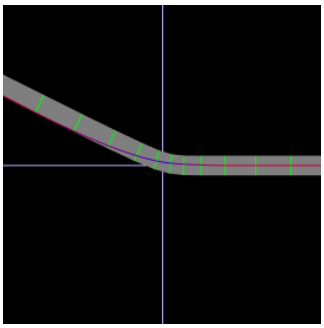
Even though the system has the ability to control the brake and throttle in this experiment, it performs worse than the best genome in the constant speed with curvature experiment. After 1000 generations the constant speed with curvature experiment presents an almost optimal race line for the constant speed. While the best genome in this experiment performs a lap, it does so by simply driving along the track mid line. Although the race lines are significantly different, the lap times are not. Compared to 415 seconds in constant speed with curvature, this experiment performs a lap by 533 seconds, which can be seen by comparing generation 1000 in table 4.2 and table 4.3.

Since the system is capable of finding close to optimal race lines when it only controls the steering, as shown in 4.1.2. It is clear that the performance is altered when the system is given the ability to control the speed. The increase in outputs and possible actions, creates a much more complex environment than in the previous experiments. The increase in complexity leads to a worse overall performance.

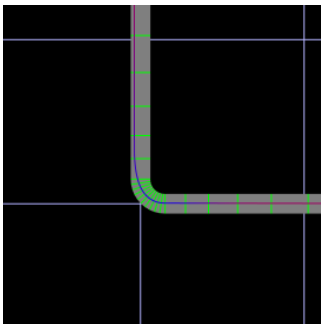
The system never stops positioning the car in the middle of the track, this suggests that the system is over trained before it is allowed to optimise its behaviour for time. Thus, the race line is hard to modify once the system can perform a lap. This initial race line is non-optimal, and thus the maximum speed that is allowed is non-optimal as well. This indicates that it might be beneficial to research the possibility of optimising for time before performing a complete lap.

4.3 Short Track Segment

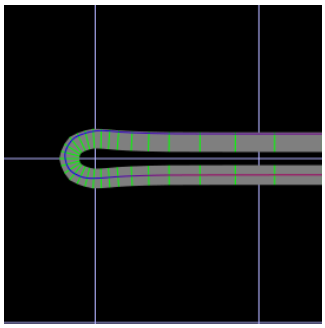
As hypothesised in section 3.4.3, the system is able to find more effective behaviours on shorter tracks than on the complete circuit. As shown in figure 4.5, the system drives at a high speed on the straights and brakes in order to steer through the corners. Thus an effective speed management behaviour, which is one of the target behaviour criteria, has been found. However, on these tracks the other criteria, the positioning, is missing. The system does not position the car towards the outer edge or turn early in the corners.



(a) Race line going through the 30-degree corner.



(b) Race line going through the 90-degree corner.



(c) Race line going through the 180-degree corner.

Figure 4.5: Race lines through the three corner types. The grey path is the track. The race line goes from red, to blue and back to red. The colour represents the speed of the car, where red is fast and blue is slow. The tracks start in the bottom right corner, and turns right and upwards the figures.

Gen.	Fitness	Time [s]
10	1067	27.24
1000	1116	20.73
10000	1148	17.20
40000	1164	15.63

Table 4.4: The progression of fitness and lap times on the track with the 30-degree corner.

Gen.	Fitness	Time [s]
10	502	–
100	1203	157.34
1000	1835	42.37
8000	2048	25.13

Table 4.5: The progression of fitness and lap times on the track with the 90-degree corner. No lap times are shown for generations where the track was not completed.

Gen.	Fitness	Time [s]
10	519	–
100	526	–
200	1456	93.99
1000	1496	86.74
8000	1699	57.13
16000	1882	38.01

Table 4.6: The progression of fitness and lap times on the hairpin track. No lap times are shown for generations where the track was not completed.

In table 4.4, 4.5, and 4.6 the progression of lap times on the short tracks is shown. On all three tracks, the system learns to complete the track in relatively few generations. After that, the lap times are significantly improved. This is mainly due to the AI driving faster and faster on the straights. Overall, the system manages to maintain a significantly higher average pace than on the complete circuit. On the complete circuit, the AI drives with an average speed of approximately 10 m/s, as seen in table 4.3 on page 4.3. Compare this with the average speeds of approximately 50, 37, and 24 m/s achieved on the short tracks.

Even though the system did not find any near perfect behaviours, the results show that the system has a greater ability to optimise for time when training on shorter tracks, than on the complete circuit. One plausible reason behind this is, as mentioned in section 3.4.3, that it is too hard to change the behaviour of the genomes once they are able to complete the longer circuit. In order to change from a cautious behaviour to a more effective one, the system must learn to accelerate on straights and brake before corners at the same time. Changing only one aspect at a time will most likely lead to a worse behaviour. If a genome is changed to accelerating more it will likely crash in a sharp corner. Likewise, if a genome is changed to braking before corners without accelerating more it will drive slower and thus decrease in fitness.

In order to further improve the behaviour, the system should learn to prioritise the positioning over the braking. Instead of positioning the car effectively and keeping the highest possible speed through the corners, the system disregards the positioning and brakes as much as is needed. How could the training process be designed to promote effective positioning behaviours? Since the system learns to

steer effectively in the constant speed experiment, one possible solution is to set a lower bound on the allowed speeds. This would force the system to position the car effectively in order to complete corners. However, deciding which lower speed bound is appropriate is problematic, since it depends on the shape of the track.

4.4 Mirrored Track

The mirrored track experiment yielded several interesting results. The test population was trained on the normal circuit until the best specimens managed to complete the circuit in approximately 500 seconds, which took 3527 generations. That population was then transferred to and trained on the mirrored track. On the first try, some specimens managed to stay on track for 1800 metres, which is about a third of the way around the circuit. This shows that at least some of the genomes had some general knowledge about how to drive. However, in the next few generations the migrated population quickly adapted to the new circuit. After only a few generations it managed to complete the circuit. This is, as shown in figure 4.6 and table 4.7, significantly faster than the control population which took approximately 150 generations to complete the circuit.

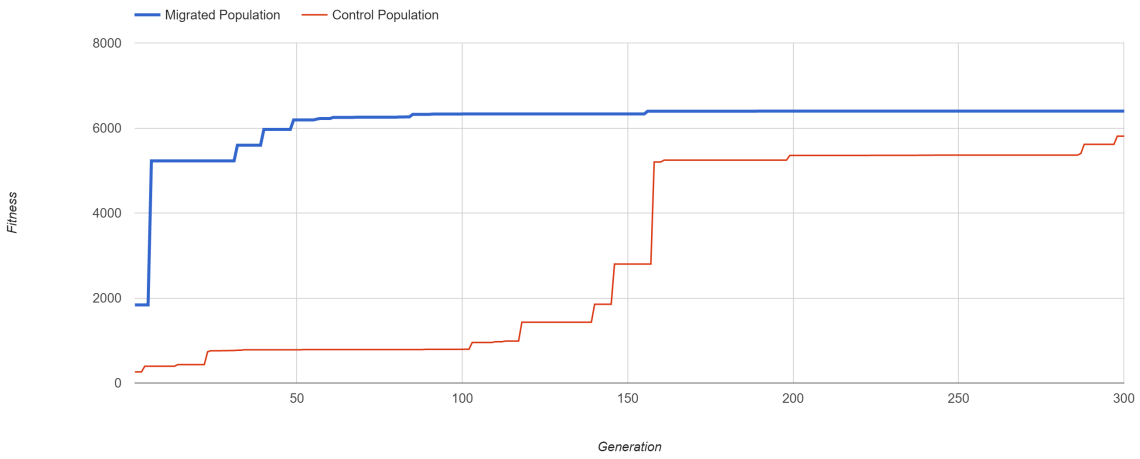


Figure 4.6: Showing the fitness progression of the migrated population shown in the thick blue line, and the control population shown in the thin red line. Note that the migrated population in a few generations surpasses a fitness-value of 5200 as a result of completing the circuit.

These results show that some of the knowledge acquired on the first track is applicable on the second. The reason may be that the genomes are adapted to similar corners on the first circuit or that the algorithm has found a network structure that is useful in a wide range of situations. In either case, one could argue that in order to find a generalised behaviour the AI should train in an environment where it encounters a wide range of situations, which in the racing domain would be different corners or combinations of consecutive corners.

Generation	Migrated Population Fitness	Control Population Fitness
1	1839	261
10	5225	397
100	6330	793
200	6397	5355
600	6484	6042

Table 4.7: Table showing how the migrated population adapted to the mirrored track compared to a control population. Fitness levels over 5200 means that the system has completed the circuit.

The extension to the experiment where the migrated population was transferred back to the original circuit once it had adapted also showed interesting results. The population forgets knowledge while adapting to the new environment. This is somewhat expected since the population mutated for 600 generations on the mirrored track, ergo no genomes trained on the original circuit remain in the population. However not all of the knowledge is lost in the process, some knowledge remains. When migrated back to the original track, one specimen manages to stay on the track for circa 2800 metres, which is slightly past halfway around. After only 5 generations of adapting back to the original track one specimen manages to complete the circuit, albeit slowly. The adaption to the original circuit is similar in nature to the adaption from the original to the mirrored one. However, one difference is that the adaption rate is slower. As shown in table 4.8, the migrated population quickly manages to complete the circuit but the improvement of lap times is slow. The migrated population improves slower than when it was trained the first time. One possible reason is that the genomes are larger and thus less flexible.

Generation	Migrated Population Fitness	Original Fitness Progression
1	2734	261
10	5264	397
100	5264	5347
500	5604	6265

Table 4.8: Table showing how the migrated population performed when transferred back to the original track. The adaption is compared to how the population progressed when originally trained on the circuit. Fitness levels over 5200 means that the system has completed the circuit.

These results show that the system is able to find generalised knowledge to some extent. However, the results also show that some knowledge acquired on the original circuit is lost during the training on the mirrored one. One possible reason is that the neural networks become over-fitted to the track they are trained on. This means that the networks have adapted to the track with behaviours that are well suited for that circuit but not in general. The traits responsible for those behaviours are then disabled or changed through mutations when the networks are adapted to the

new environment, resulting in a loss of knowledge. One could argue that this loss of knowledge is desired, since the loss of knowledge is a part of adapting to the new environment. However, it is also desirable if the system can learn to solve new problem while it continues to be effective on the earlier problems. If the system stops being evaluated in a specific environment there is no mechanism that preserves knowledge specific to that environment. Thus in order for the system to remember how to solve a problem, the evaluation on that problem must continue.

4.5 Discussion on the Physics Simulation

The speed held for the experiments was generally low in comparison to the capabilities of the car. The speed achieved, at least for the large track, was about 10 m/s whereas the maximum speed allowed was almost 100 m/s. The constants for the car performance was inspired by formula 1 cars but one mistake was observed late into the experimentation process. The force used for the maximum centripetal force F_c used to turn and rotate the car was notably low, a tenth of the maximum deceleration force. Edmondson (2011) present measurement data for a high performance sports car, for which the maximum lateral acceleration was about double the maximum deceleration. However, it was conceived that the error would only effect the speed for which different optimal behaviours occur, and not the conceptual characteristics.

On the other hand, if the possible speed for which corners could be taken would be larger, the feasible range of speeds would also be larger and that might make it easier for the controller. It might also be worth investigating whether an increased possible speed through the corners, that would close in slightly to the maximum speed, also would make it easier to find a more distinguished behaviour on the straights.

Concerning some of the aspects of physics that were ignored, it probably was a good choice as the results for the simplified physics was not close to optimal. Early experimentation showed that a combined traction budget for the two axis of forces, increased the training times and reduced the performance, understandably as the complexity of the problem increased. This is probably also true for other neglected phenomena.

4.6 Discussion on the NEAT Algorithm

As described in 2.2.7, the NEAT algorithm is detached from the actual problem it solves. It is only responsible for the evolutionary process. No analysis of the problem or the simulation results is done in the NEAT system. The only feedback to the system is the fitness value. This makes the algorithm flexible, it is easily adapted to new problems since it is domain agnostic. One might however question how effective it is when solving complex problems.

The algorithm is not smart. It does not know or understand what aspects of the problem it excels at or what behaviours it is lacking. Therefore, the algorithm has no possibility of applying directed modifications. The progression of the algorithm

depends on the availability of an easily reachable better solution. The network topologies are gradually constructed. Each genome is mutated a limited number of times each generation. Thus larger changes to a genome take several generations. In order to produce a successful genome, it is desirable if each gene in the genome is beneficial on its own. If a combination of genes is beneficial but some of the individual genes negatively affect the performance on their own, it is likely that the genes will be discarded before a genome with the combination is found.

If a network performs badly in comparison to the other members of the same species, or if a species performs significantly worse than the other species they will be discarded[21]. Additionally, if a species has stagnated, which means that it has not improved for a specified number of generations, it will also be discarded. A third aspect is that worse performing species will have fewer children, thus shrinking in size and in potential diversity. Therefore, the range of time where individual genomes and new innovations can survive is rather well defined. In order to survive it is necessary to improve or at least maintain an above average fitness. Thus it is important for genomes to stay functioning when mutated. Mutations that lead to a significantly worse fitness will fall out of the favourable range and get discarded. It is therefore important that the complexity is gradually increased. If an improvement requires several mutations, NEAT may not be able to find the correct combination of mutations. Thus if NEAT is successful, it likely progressed by a series of small beneficial mutations.

One example of this is results shown in section 4.2. The genomes learn to drive slowly in order to complete the circuit. Then they will be compared on their lap times, but they may be somewhat locked into the behaviour learnt to complete the circuit. It is possible that there does not exist a path of gradual beneficial mutations to a more aggressive and effective behaviour. The genomes may be stuck in a local fitness maximum. This raised the question of whether it is better to complete the full circuit slowly or only completing a shorter segment but with an effective behaviour

The difficulty of gradual progression may be particularly problematic when achieving behaviours requiring a combination of several components that require each other to be beneficial. If several aspects are required to progress simultaneously, and any of them are missing, it might cause a failure. One example of this can be seen in the full control experiment. If a network changed to drive at a higher speed, but does not change the position of the car accordingly it will likely cause the car to crash. This fact might be one of the reasons as to why some experiments do not perform in optimal ways.

NEAT has proven itself to be useful. We can see in some experiments, e.g. shortest path, that the algorithm is able to find beneficial behaviours that often resemble target characteristics. But it is seldom able to learn all of the smaller details. A behaviour with all of the target characteristics has not been found. Thus it seems like it is hard to find the optimal behaviour using NEAT, due to the problems mentioned above. However, these observations on the key mechanics of the algorithm and the results in our study does not prove that NEAT is unable to find a general and optimal behaviour. Though the observations highlight a structural weakness that cannot be denied.

4.7 Problem Modelling

In order to achieve the best possible results, it is important to utilise NEAT to its fullest. One important aspect, is how the problem is modelled for the neural network. This includes what information the neural network is provided, and how the results are interpreted by the network.

The decision process can be divided into two steps. The first is to process and interpret the provided data. The second is to make a decision based on the processed information. It seems like it is beneficial to let the networks focus on the second step. This can be done by only feeding the neural networks with the information required to make good decisions.

A way to analyse different sets of data is by the information they contain and how relevant it is to the problem. If the data contain noise the training have a more difficult task to learn.

One example is how the system sees the track. The networks could be provided with information describing the track as it is represented in the simulator, a set of three dimensional vectors defining a set of triangles. However, it would have to learn how to interpret and correlate the different data points. The curvature data describes approximately the same thing, which a significantly lower amount of data. The exact coordinates of the triangles are not relevant, the important information is the shape of the track relative to the car.

To some level, the developer need to do the analysis of what is relevant and good information. If the data get to compact or hard to calculate with, it might get difficult to learn to use it, based on the discussion about multiple goals in section 4.6.

To some extent it might also be important that the values are easily calculable, to make it simpler to find a usage. Several of the input data that was repeatedly used did not provide new information, but was other representations of data already provided. Distance to middle, right and left edge are in a sense analogous as the track in our experiments was uniform in width. The curvature segment data are simply the summation of some of the other data points.

We do often observe that both variants, distances and curvature and their transformations, are used at the same time. It suggests that both variants were usable in the training process, at least at some point. They could have been calculated from the other form of data, but that would have required a more complex network.

4.8 NEAT Usability

The results presented contain both aspects of success and failure. The algorithm manages the control tasks to some degree, but does not achieve the target behaviour, when presented with more complex problems. As discussed in section 4.6, the performance is limited when the steps in which it is required to progress are too large.

As described in section 2.2.7, NEAT is not an algorithm designed to solve a particular problem. Instead it produces artificial neural networks through neuroevolution. The process is solely controlled by the fitness value. As such, NEAT do not

do any particular analysis of the actual problem. If NEAT is sufficient to solve a problem, the simplicity makes the algorithm easily implementable as few domain specific processes are needed. It also means that the developer is not required to have as deep domain knowledge, as the algorithm managed to solve the problem on its own. It usually requires more effort and advanced knowledge in order to solve a problem, than it is to know about the problem.

A different approach is as was done in the Stanley project, where not one entity in the system controlled all aspects the driving task [2]. Different parts of the system were responsible to handle aspects such as perception, planning and steering. Also, in The 2009 Simulated Car Racing Championship the most successful solution had different components in its controller [26]. In a similar way, NEAT might be used with greater success if the particular neural networks are trained for more focused purposes.

5
Conclusion

The project yielded several interesting results as well as insight into how to utilise NEAT to its fullest potential. Applying NEAT to the racing domain have proven to be a complex problem. However, NEAT is capable of creating ANNs that can drive with some of the target behaviours.

5.1 Racing

The overall results of these experiments show that NEAT have potential in the racing domain. It manages to find behaviours for both positioning and speed management, though not simultaneously.

When the training had the focus of only steering the car, some planning behaviours were apparent. Prior to a corner, the car would steer out towards the edge of the track. Therefore, giving itself a better position and the ability to complete the corner. This behaviour gave the car the ability to complete the track at constant speeds that gave larger turning radius than some corners of the track.

At tracks consisting of a single corner in between two straights, the car managed to control and plan the speed prior to the corner. It managed to accelerate at the straights, and decelerate to a speed low enough to complete the upcoming corner. Though, indications of positioning to handle a larger turning radius have not been apparent in these experiments.

When the system has been able to control all aspects of the car, the absence of position planning has not yielded the complex behaviour of a racing driver. To first focus on making the car to take a compete lap, and after that, focus on driving as fast as possible, influences the behaviour significantly. To portray both of these behaviour at the same time, the fitness function handed to the NEAT algorithm would need to give both of these aspects the same priority, in difference to what have been made in these experiments.

5.2 NEAT

The fitness function used greatly affects the behaviour and the progression of the system. In order to utilise the algorithm to its full potential, it is essential to find a suitable fitness function for the problem domain. It is also important that the algorithm is able to progress smoothly, with a gradually increasing fitness. An increase in fitness should correlate to an improvement of the systems behaviour. If an improvement leads to a decrease in fitness, the system might get stuck in a local fitness

maximum. Additionally, NEAT is only able to make a limited number of modifications to each genome in each generation. Thus large changes to a genome might take several generations to implement. This property highlights the importance of improving by a series of small individually beneficial modifications.

6
Future work

As discussed in section 4.6, the success of NEAT relies heavily on the fact that it is able to progress in small steps towards a solution. In this project, the possibility of changing the fitness function as the AIs performance increases was examined. It would be interesting to change the problem in a similar way. Possible experiments include changing tracks or increasing the limit of how far the car is allowed to drive, as the AIs performance increase.

The set of input and outputs used during the project is rather small. Further researching possible representations might lead to major improvements in performance. One particular set of inputs that deserve particular attention is the representation of curvature. With the current curve representation, a change in timing to perform some action, for example approaching the tracks right edge 300 meters before a corner, instead of 100 meters before a corner requires a change in network topology. Changing topology is problematic, since large amount of training is required to perform major topology modifications. One potential alternative to the current model, may be to represent the upcoming corners as groups of inputs. Where one group might consist of distance to, the angle or shape of, and length of the corner.

The results show that the complexity becomes too large when introducing the speed control output. It seems reasonable to introduce more networks in order to divide the problem into several tasks. One approach may be to divide the tasks into two, steering and controlling the speed. It is also interesting to consider classification of corners. This could potentially be achieved by training networks to perform certain corners, and then using a classification AI to pick a network that determines the behaviour. This is particularly interesting, since results also show that networks perform well when trained on individual corners instead of a complete track.

It has been shown that modularity can be evolved by neuroevolution techniques when the training process alternate between environments that take up different parts of the input data set [27]. However, this differs to the racing solution presented in this report as the same input variables are used regardless of environment. It is worth investigating if the same modularisation tendency is achievable even for this data model. Possibly, structure that enable such modules is required for it to work, which would add considerable complexity to the network.

It would be interesting to consider the possibility of replacing NEAT with another machine learning algorithm. One type of reinforcement learning that has shown great potential lately in continuous action spaces is deep reinforcement learning [28]. This could potentially solve some of the problems that were encountered during the project. Furthermore, it would be interesting to see how the algorithm

performs in comparison to NEAT.

Bibliography

[1] D. Stavens, *Learning to Drive: Perception for Autonomous Cars*. PhD thesis, Stanford University, http://purl.stanford.edu/pb661px9942, 5 2011.

[2] S. Thrun *et al.*, "Winning the darpa grand challenge," *Journal of Field Robotics*, 2006.

[3] B. Huval *et al.*, "An empirical evaluation of deep learning on highway driving," *arXiv preprint arXiv:1504.01716*, 2015.

[4] "Google AI algorithm masters ancient game of go," *Nature*, vol. 529, p. 445–446, January 2016.

[5] "Alphago." https://deepmind.com/alpha-go.html. Accessed: 2016-05-10.

[6] "Roborace Announcement." http://fiaformulae.com/en/news/2015/november/formula-e-kinetik-announce-roborace-a-global-driverless-championship.aspx. Accessed: 2016-05-05.

[7] N. Gkikas, "Formula 1 steering wheels a story of ergonomics," *Ergonomics in Design: The Quarterly of Human Factors Applications*, vol. 19, no. 3, pp. 30–34, 2011.

[8] World Health Organization, "Global status report on road safety 2013: supporting a decade of action." http://www.who.int/violence_injury_prevention/road_safety_status/2013/en/, 2013.

[9] J. Fuglestvedt, T. Berntsen, G. Myhre, K. Rypdal, and R. B. Skeie, "Climate forcing from the transport sectors," *Proceedings of the National Academy of Sciences*, vol. 105, no. 2, pp. 454–458, 2008.

[10] J. M. Anderson, K. Nidhi, K. D. Stanley, P. Sorensen, C. Samaras, and O. A. Oluwatola, *Autonomous vehicle technology: A guide for policymakers*. Rand Corporation, 2014.

[11] "The official site of TORCS, the open racing simulator." http://torcs.sourceforge.net/. Accessed: 2016-06-01.

[12] B. Beckman, *The Physics of Racing*. 1991. http://phors.locost7.info/contents.htm, Accessed 2016-05-09.

[13] C. Edmondson, *Fast car physics*. The Johns Hopkins University Press, 2011.

[14] R. K. . F. Provost, "Glossary of terms - special issue on applications of machine learning and the knowledge discovery process," *Machine Learning*, no. 30, pp. 271–274, 1998.

[15] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Pearson, 2 ed., 1999.

[16] T. M. Mitchel, *Machine learning*, p. 105. McGraw-Hill Science/Engineering/-Math, international edition ed., 1997.

[17] D. S. Whiteson, "Adaptive representations for reinforcement learning," 2010.

[18] W. D. Smart and L. P. Kaelbling, "Practical reinforcement learning in continuous spaces," in *ICML*, pp. 903–910, Citeseer, 2000.

[19] A. L. Nelson, G. J. Barlow, and L. Doitsidis, "Fitness functions in evolutionary robotics: A survey and analysis," *Robotics and Autonomous Systems*, vol. 57, no. 4, pp. 345–370, 2009.

[20] F. Gomez, J. Schmidhuber, and R. Miikkulainen, "Accelerated neural evolution through cooperatively coevolved synapses," *The Journal of Machine Learning Research*, vol. 9, pp. 937–965, June 2008.

[21] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[22] F. Gomez, J. Schmidhuber, and R. Miikkulainen, "Efficient non-linear control through neuroevolution," in *Proceedings of the European Conference on Machine Learning*, (Berlin), pp. 654–662, Springer, 2006.

[23] "F1 rules & regulations: Scrutineering and weighing." https://www.formula1.com/content/fom-website/en/championship/inside-f1/rules-regs/Scrutineering_and_weighing.html. Accessed: 2016-05-09.

[24] "NEAT C++ implementation." http://nn.cs.utexas.edu/?neat-c. Accessed: 2016-05-02.

[25] "Super MarI/O Lua implementation." http://pastebin.com/ZZmSNaHX. Accessed: 2016-05-02.

[26] D. Loiacono *et al.*, "The 2009 simulated car racing championship," *IEEE Transactions on computational intelligence and AI in games*, vol. 2, pp. 131–147, June 2010.

[27] K. O. Ellefsen, J.-B. Mouret, and J. Clune, "Neural modularity helps organisms evolve to learn new skills without forgetting old skills," *PLoS Comput Biol*, vol. 11, no. 4, p. e1004128, 2015.

[28] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

A

Project Settings

During the project settings have been determined for different parts of the implementations. These include probabilities for network mutations as well as constants that determine the importance of lap times in the fitness function.

This appendix presents the configurations that were used during the course of the project.

A.1 NEAT Constants

Setting	Value
Delta Disjoint	2.0
Delta Weights	0.4
Delta Threshold	1.0
Population	500
Staleness Tolerance	20
Step Size	0.1
Mutate Connections Chance	70%
Perturb Chance	90%
Crossover Chance	75%
Link Mutation Chance	50%
Node Mutation Chance	40%
Bias Mutation Chance	40%
Disable Mutation Chance	40%
Enable Mutation Chance	20%

A.2 Fitness Function Constants

Setting	Value
λ	5