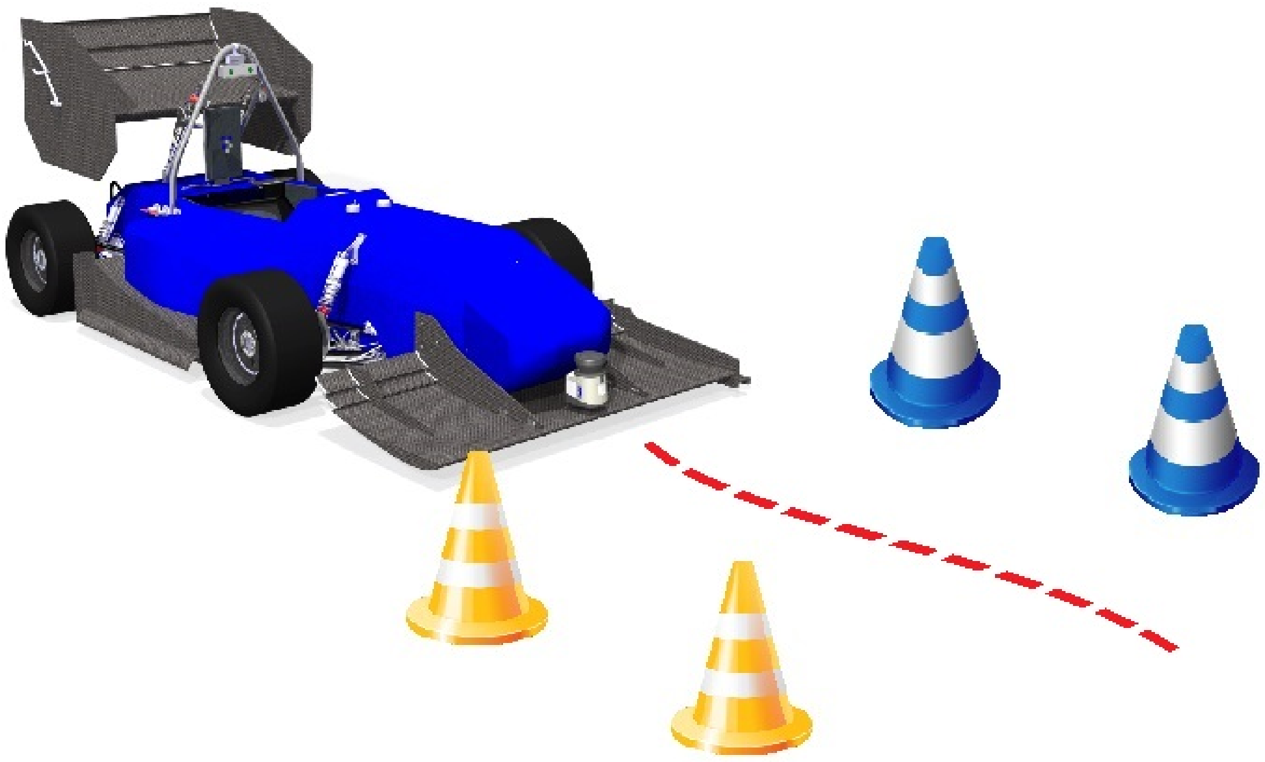
CHALMERS



Autonomous vehicle control: Exploring driver modelling conventions by implementation of neuroevolutionary and knowledge-based algorithms

Master's thesis in Systems, Control & Mechatronics and Complex Adaptive Systems

LINUS ARNÖ
JONAS ERIKSSON

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2018

MASTER'S THESIS IN SYSTEMS, CONTROL & MECHATRONICS AND COMPLEX ADAPTIVE SYSTEMS

Autonomous vehicle control: Exploring driver modelling conventions by implementation of neuroevolutionary and knowledge-based algorithms

LINUS ARNÖ
JONAS ERIKSSON

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2018

Autonomous vehicle control: Exploring driver modelling conventions by implementation of neuroevolutionary and knowledge-based algorithms
LINUS ARNÖ
JONAS ERIKSSON

Master's thesis 2018:83
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
A Formula Student race car tasked with generating suitable commands for moving between the cones.

Chalmers Reproservice
Göteborg, Sweden 2018

Autonomous vehicle control: Exploring driver modelling conventions by implementation of neuroevolutionary and knowledge-based algorithms
Master's thesis in Systems, Control & Mechatronics and Complex Adaptive Systems
LINUS ARNÖ
JONAS ERIKSSON
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

Abstract

In this paper an investigation of driver modelling conventions is presented. The goal was to compare traditional driver modelling with machine learning, to find indications of when one approach could be preferred over the other. This was done by implementing some representatives of the different approaches and evaluating them in the same conditions. The traditional approach was represented with one well established model by Sharp et al., as well as one self made aim point model. Both of these required a path planner and velocity control, that were also designed by the authors themselves. The machine learning approach was represented by neuroevolution, an alternative technique for solving reinforcement learning problems, and specifically the method called NEAT. The results showed that all implemented methods were able to solve the task, but in the specific scenario and with the current amount of training the two traditional models were superior to the evolved neural network. Similarities and potential reasons for differences between the models are discussed, as well as some identified advantages and disadvantages to both approaches.

Keywords: Autonomous control, driver modelling, motion planning, neural networks, neuroevolution

Preface

In 2017, Chalmers University of Technology initiated the project Chalmers Formula Student Driverless (CFSD) to provide students with hands-on experience from developing autonomous driving systems. The objective of the project is to participate in Formula Student (FS), Europe's most established educational engineering competition, where students compete in designing and manufacturing race cars. CFSD will specifically participate in a branch of the competition created in 2017, called FS Driverless, where the FS cars are to race autonomously. The authors are a part of the CFSD team currently consisting of twelve master students that will transform a former built FS race car to drive autonomously through three different tracks marked by cones. The results of this thesis will be considered in the CFSD project.

Acknowledgements

We would like to thank our supervisor Ola Benderius for the guidance and help in structuring the entire project, and of course for taking the time to evaluate and leaving constructive feedback of our work.

We would also like to thank the Revere team for sharing their facilities, helping out when there were technical difficulties and for being good company.

Last but not least, we would of course like to thank the entire CFSD18 team for the great cooperation and fellowship. This thesis would not have happened without the CFSD project and the dedication of everyone involved.

Thesis advisor: Ola Benderius
Thesis examiner: Ola Benderius

Nomenclature

ADS	Autonomous Driving System
AI	Artificial Intelligence
ANN	Artificial Neural Network
CFSD	Chalmers Formula Student Driverless
CFS17	Chalmers Formula Student 2017
EA	Evolutionary Algorithm
FS	Formula Student
FSG	Formula Student Germany
GA	Genetic Algorithm
NEAT	NeuroEvolution of Augmenting Topologies
TWEANN	Topology and Weight Evolving Artificial Neural Network

Contents

1 Introduction

Transportation, as we see it today, is standing in front of a fundamental change. The automotive industry together with universities and high-tech companies world wide are spending vast resources on developing the necessary technology to make fully autonomous driving available to the world. The underlying reasons for the effort are many. Autonomous vehicles are believed to result in fewer accidents, lower emissions, and more efficient use of infrastructure [1].

A human driver can be seen as a part of a closed-loop system consisting of the surrounding environment, the driver and the vehicle. Autonomous driving systems (ADS) are commonly designed as artificial human drivers. Comparable to the human eyes, a perception system is used to collect information about the environment. The information is sent as input to a decision process, or *brain*, which determines desired actions. The actions are then executed by a mechanical system that turns the wheels or locks the brakes. Driver behaviour, i.e. the output to vehicle actuation given an environmental input, could be argued to determine the skill of a driver or ADS, provided functioning perceptive and mechanical systems. In an ADS, the decision how to brake, steer or accelerate is made by a driver model. Driver models have long been utilized in research to understand driving as a phenomenon, but have been found relevant in several practical applications such as design of infrastructure, vehicles and active safety systems [2]. They were originally used to simulate human drivers in computer simulations, but have gained increased usage as integral parts of embedded real-time systems.

In the past decades there have been several approaches to driver modelling. Examples are the well-known model by Sharp et al. [3] which is based on control theory and focused on following a desired drive path as closely as possible, and the Salvucci and Gray model [4] that uses assumptions about human driving behaviour without need of an internally planned path [2]. What these models have in common is that they determine driver actions based on rules decided by the designer and motivated by solutions from previous research. Such driver models could be referred to as knowledge-based. Knowing the model, it is possible to predict the output given a certain input. This as opposed to systems utilizing Artificial Neural Networks (ANN), which has proven to be a powerful tool in the field of autonomous driving [5].

With breakthroughs in research and increasing access to more processing power, deep learning has drastically improved computer image recognition and is commonly used in perception systems for autonomous vehicles. However, to solve the vehicle control task using ANNs is an ongoing research area. In 2016, based on the findings of Pomerleau [6], Bojarski et al. [7] trained a convolutional neural network (CNN) to map only the raw pixels from a front-facing camera to the steering command of a vehicle. The system, DAVE, was demonstrated the same year in NVIDIA's self-driving car [8]. This was achieved using a technique called *behavioural cloning*, where a network was trained from human driving data. Other promising techniques involve different approaches to *reinforcement learning* [9]. In these deep learning techniques, the topology of an ANN i.e. the structure of the neurons and connections, is designed by a human based on some level of experience [10]. The natural neural network in a human brain is however a result of millions of years of evolution. Interestingly, evolution of brains in nature is the only known example of a process that has created something truly intelligent. This is what drives the AI branch of neuroevolution, where the goal is to trigger an evolutionary process that creates intelligence inside a computer. Neuroevolution combines evolutionary algorithms and ANNs, and have in recent studies proven to be a competitive alternative when solving reinforcement learning problems [11]. An interesting topic within neuroevolution is *topology and weight evolving artificial neural networks*, TWEANNs, where not only the weights of a network are trained using evolutionary algorithms, but also the full structure of the network. It is a process much resembling the evolution of intelligence seen in nature.

1.1 Purpose

The traditional approach to autonomous driving is knowledge-based, where each sub-task is solved with elaborate and well motivated designs by the developers. It is favored for its modular characteristics, where each sub-component can be substituted and interpreted individually. A different approach that is becoming more and more popular is to instead make use of artificial neural networks, favored for the potential of overcoming problems that are too complex for humans. However, its black-box characteristics result in a loss of modularity that makes it difficult to interpret the system and understand how and why decisions are made and how to correct undesired behaviour. Researchers are divided between these approaches, and comparisons of strengths and weaknesses would be helpful for deciding how future research efforts should be distributed. The purpose of this thesis is to provide a basis for finding indications for deciding when one approach is to be preferred over the other.

This project has focused on the driver model, i.e. how to map the information from the perception system into appropriate driving actions. The goal was to compare the two mentioned driver model conventions by implementation and simulations. To achieve a larger contrast, a new full-insight model was designed and together with a well established knowledge-based model compared against a no-insight ANN. The ANN was developed using a neuroevolutionary TWEANN, automatically generated based on performance rather than conscious design decisions. Knowledge-based and TWEANN driver models are on opposite sides of the human-machine intelligence spectrum, where one is a white-box intelligence designed by a human, and the other a black-box intelligence evolved in a computer with minimal prior knowledge added by humans. The conventions were evaluated in simulated scenarios resembling an FS competition event. Specifically, the work has been focused towards answering the following research questions:

RQ1: How well do knowledge based driver models and a black-box artificial neural network perform compared to each other?
RQ2: Which one of the methods is the most flexible to vehicle specific and environmental changes?

1.2 Limitations

The driver models have only been evaluated in FS competition scenarios. This introduces simplifications compared to real world traffic conditions. These include single lane following without intersections or obstacles such as other road users. The lane is also marked with sparse point markings rather than conventional road lines. An assumption is made that the perception system is ideal and consistently provides a given number of the points that mark the drivable surface. The comparison has been made between three driver models. It was not sought to design or implement the best possible performing models of the represented fields, nor to achieve the highest potential of the implemented models. The objective was merely to explore the approaches that are about to compete for resources of research within autonomous driving in the upcoming decade. Thus, the knowledge-based models were tuned heuristically and the network evolution was performed with a default parameter setting not proven to be optimal for the specific task. Also, the effort spent on discovering the optimal training set up was limited. Furthermore, the comparison did not aim to reach a high level of resemblance to real world scenarios. Therefore, a simplistic vehicle model was used to provide the necessary platform for the comparison.

2 Theory

This chapter begins by introducing traditional driver modelling, where a selection of existing driver models are presented. These served as references and inspiration for the design of the aim point model, introduced in Section 3.2, as well as candidates for representing the traditional driver models. This is followed by a brief presentation of the Sharp model, which was chosen as a well established knowledge-based driver model to implement for comparison. Furthermore, some neuroevolutionary techniques for creating and evolving TWEANNs are introduced before the theory of the chosen method is presented more thoroughly.

2.1 Knowledge-based driver models

Traditional driver models are in most cases focused either on lateral or longitudinal control. Thus, limiting the scope and narrowing down the area of research. To enable modularity is also preferable in embedded systems. Regarding lateral control, the literature dates back to 1947 where the first mathematical model of lateral driver steering control was published by Tustin [12]. The model approximated driver control by a linear transfer function with an additional non-linear part that was referred to as the remnant, of which several attempts has been made to explain in a model. In 1953, Kondo developed the first model of driver steering behaviour using a preview point [13]. The preview point was then referred to as the sight point related to where the driver is looking. The distance between the driver and the preview point was furthermore hypothesized to have a linear relationship with the vehicle speed. The angular error between vehicle heading and preview point can be used as a minimization measure that later was recognized to minimize both lateral error and heading error [14].

The concept of using preview points can be found in most modern driver models. In the well-known model by Sharp et. al. (2000) [3] the concept is further developed to compute the steering angle from a weighted sum of deviations from several points on a preview distance. In 2004, Salvucci & Gray formulated the *two-point visual control model of steering* [4], which is rather mathematically similar to the Sharp model but applies control by minimizing the movement of a near and a far point on a target lateral position, and the angle to the near point. The model was developed after Land and Horwood's model [15] that argues that drivers use information from a far region where the driver is looking, and a near peripherally perceived region. Another well-known model is by MacAdam (1981) [16], which aims to minimize the squared deviation error from a path using different optimization criteria. Similar to the models of Sharp and Salvucci & Gray, MacAdam's model is based on following a predefined path. A different branch is *satisficing* driver models, which are applying steering corrections on compelling need rather than any small error [17]. The model of Gordon & Magnuski (2006) [18], is an example where the vehicle is controlled to stay inside a region modelled using boundary points. The current yaw rate is compared to the one needed to avoid the boundary point with the largest mismatch.

When discussing lateral driver models, it may be useful to categorize models according to their structure. In 2012, Benderius defined three different design perspectives: a behaviour perspective, a cognitive perspective and a control perspective [19]. A behaviour perspective is most suitable when designing models based on observed or hypothesized driver behaviour. The cognitive perspective uses models of the human mind as framework, however, since little is known about how the human mind works it is commonly used with specific driver phenomena seen in traffic. In this thesis, little emphasis is made on how well a model mimics human driving behaviour, more how to follow a high curvature path at highest possible speed. It is therefore interesting to look at driver models from a control perspective which typically are focused on following a desired path as closely as possible. From this perspective, the driver is considered a controller with the vehicle as plant. Seeing the driver as a controller allows methods from control theory to be applied directly to the model. As an example, an approach would be using optimal control methods with a cost function that minimizes deviations from a given path, as was done by MacAdam and Sharp et. al. A common approach used for both lateral and longitudinal control within autonomous driving is using *model predictive control* (MPC) techniques [20][21][22][23]. However, such approaches demand an accurate vehicle model to predict future response of the system. Since the performance of such a model would rely heavily on the accuracy of a vehicle model, it is not suited for the comparison to be made in this thesis.

Speed references for longitudinal control in real life traffic situations are often based on speed limits or on-road obstacles such as other road users. This thesis considers unlimited obstacle free lane following, meaning that speed planning is based on road properties such as road friction and curvature, together with vehicle dynamic properties. Road friction is difficult to estimate, therefore the curvature of the desired path plays a large role in speed planning. Paths with continuous upper bounded curvatures are beneficial for smooth path following since they minimize control-input variability and might require less control effort to track depending on the controller. Fraichard & Scheuer [24] used a set of optimal curves (straight line segments, arcs of circles, and clothoids) to implement an algorithm based on Dubins' curves [25]. Simple turns are modified into continuous-curvature turns, created with aid of clothoids. However, between circular arcs and straight line segments generated by clothoids, the path presents discontinuities. Several researchers have therefore proposed alternative fundamental curves, such as B-splines [26], quintic polynomials [27], polar splines [28], cardioids [29], G2-splines [30], η^3-splines [31], and Bézier curves [32]. These curves all have coordinates with closed-form expressions, unlike for clothoids where approximations and look-up tables are required. Knowing the curvature of the path, a speed profile consisting of maximum desired velocities can be determined using friction limits and vehicle properties. Based on the speed profile, a reference velocity may be decided and by this the required acceleration is acquired. The outputs to vehicle actuators could then be regulated using conventional control techniques. There is however not a necessity to use an continuous curvature path, which introduces uncertainties in the function fitting process and requires more computational power. In Section 3.1, more is explained about the method used in this work.

2.2 The Sharp driver model

In this section an overview of the steering model by Sharp et al. is presented, and more details can be found in the original paper [3]. A vehicle is tasked with following a known global ideal path. An *optical lever* is extended in front of the vehicle for a distance L, determined by multiplying the current vehicle speed and a set preview time. Then n points l_i are placed on the optical lever, starting with l_1 in the vehicle's center of gravity. The lateral off-sets e_i are found by placing reference points r_i on the path perpendicular to the l_i points, and measuring the distances between them. This means that e_1 is the vehicle's lateral off-set from the intended path position, and the following e_i are preview information errors. An additional feedback value e_Ψ is used, representing the angular difference between the vehicle global heading, and the tangent angle of the intended path position. Finally the control gains K_i are introduced to calculate the desired steering angle δ according to equation (2.1). Exponentially decreasing control gains have been found to be a good basis, but should be further tuned until a satisfactory behaviour is achieved. A picture of the model is shown in Figure 2.1.

$$\delta = K_\Psi e_\Psi + K_1 e_1 + \sum_{i=2}^{n} K_i e_i \tag{2.1}$$

2.3 Neuroevolution

Evolutionary algorithms (EA) is an umbrella term for stochastic optimization algorithms inspired by natural evolution [33]. EA uses methods that resembles natural selection, reproduction and mutation. A population of different solution strategies are evaluated for the problem at hand, and the most successful individuals have a higher chance of contributing to the strategies that will form the next generation of the population. Iterating the procedure for many generations will result in individuals carrying combinations of the most successful strategies from former populations, and should eventually lead to at least one individual with a close to optimal solution. A common type of EA is the *genetic algorithm* (GA), where solution strategies are encoded similarly to human chromosomes. The reproduction usually closely resembles biological methods, with offspring being generated by crossover between two parent chromosomes and sometimes a few mutations of individual genes.

In neuroevolution, an EA procedure is combined with ANNs to find high performance networks for specified tasks. Recent findings [11][34][35][36][37] has brought back attention to neuroevolution, as it has been shown that it can compete in complex problems that have previously favored reinforcement learning techniques. A network with evolutionary optimization of both structure and connection strength is referred to as a TWEANN, and there are multiple methods to create one. One of the earliest methods for constructing and training recurrent neural networks was *generalized acquisition of recurrent links* (GNARL) published in 1994 [38]. The original paper introduces the method by presenting its performance in a couple of problems of interest. It also

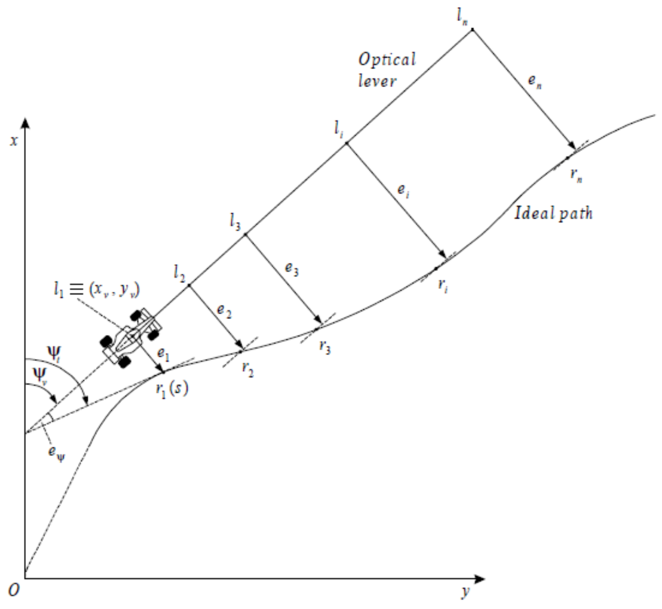


Figure 2.1: Visualization of the Sharp steering model. Picture from [3].

makes an argument for why conventional GAs are not suitable for evolving ANNs. The main reason being that the standard dual representation of the search space and the dynamics of crossover do not translate well into making flexible network topologies. Certain problems arise such as incompatible parent topologies and the *competing conventions* problem where different chromosomes can give networks with identical but rearranged components. Such problems endanger the computational ability of offsprings. The problems with normal GAs warrant the development of gene structures specifically made for ANNs.

In 2002, Stanley & Miikkulainen presented a TWEANN method called *neuroevolution of augmenting topologies* (NEAT) [39]. The method offered a new solution to the problem of genetic crossover of networks, and became popular for experiments in the field. With NEAT, the competing conventions problem was avoided by lining up the chromosomes according to the historical origin of the genes before doing the crossover, resembling the biological process of synapsis where crossed genes are ensured to encode similar traits. It also avoids premature elimination of new topologies by applying speciation. Speciation temporarily divides the population into smaller groups of structurally similar individuals, allowing new network structures to evolve within their species and get closer to their full potential before being evaluated against the full population. NEAT uses direct encoding so that the decoding of genes will not restrict the network to specific topology classes. Some years later NEAT got an extension called HyperNEAT (2009) [40] that supports indirect encoding which allows for deeper networks.

Another method focusing on network scalability is *discover & explore neural network* (DXNN) published in 2010 [41], which is solving the *curse of dimensionality* with a highly modular and hierarchical structure of connected clusters of smaller networks called sub-cores. It substitutes speciation with a two-phase process. In the tuning phase every individual is allowed to mutate until it achieves the best fitness that topology can offer in a reasonable amount of time. Natural selection is carried out in the next phase, which does not start until the tuning phase is over. The paper also presents tests of how DXNN performs in different problems and compares it with other TWEANN methods. The conclusion is that highly complex problems can often cause other methods to become bloated with newly created neurons. DXNN can usually find a relatively compact solution by requiring an increase in structural complexity to deliver an approximately equivalent increase in performance as compensation.

2.4 Neuroevolution of augmenting topologies

In this section an overview of the inner workings of NEAT is presented. It is intended as a summary rather than a complete description, more details can be found in the original paper by Stanley & Miikkulainen [39] were the following information also originates from. As the first part of the name suggests it is a neuroevolutionary technique, which means that an artificial neural network is trained with an evolutionary algorithm. As the second part of the name suggests, the network is not only trained by altering the connection weights but also altering the structure by adding new nodes and connections. The motivation behind the development of NEAT is to solve the three most serious problems with the TWEANN methods that came before it. These problems are:

The competing conventions problem: Also called the permutation problem, competing conventions is when a single neural network can be described with more than one genome encoding. This causes problems in the reproduction process of crossover. If two different encodings for the same network are combined there is a considerable risk of including some of the functionality in more than one place in the new genome, at the cost of leaving something else out entirely. This would endanger the functional validity of the offspring of two healthy parents. Furthermore, TWEANNs are especially vulnerable to competing conventions since networks representing similar functions might have completely different topologies or differently sized genomes, so that previous solutions to the competing conventions problem for fixed networks do not apply.

Protecting innovation: Mutations in TWEANNs will occasionally cause a structural change in the network. Such a change will however very likely result in an initial decrease of the individual's fitness. If only fitness is considered, innovative individuals would always be at a disadvantage compared to the rest of the population that was left unchanged in that generation. A solution is to make use of *speciation*, also known as *niching*. This would divide the population into groups of similar individuals, which allows a newly mutated structure some generations to optimize and get closer to its full potential before getting compared with the population in full. It is however not commonly used in TWEANNs because it requires a way to calculate the similarity of individuals, and as already mentioned TWEANNs with similar functionality might have radically different topologies and genomes.

Innovation of topology: It is common to start training with an initial population of random networks. In TWEANNs this would typically mean that the starting topologies would also be random to ensure starting diversity. Other than the risk of creating networks with disconnected nodes or incomplete connection paths this would also waste computation time on networks that might be too big to begin with or having nodes where they are completely unjustified. There is a computational value to finding the minimal solution, and one natural way to minimize the size of a network is to give a penalty to size in the fitness function while training. The difficulty in that is however to decide the magnitude of this penalty, since the ideal network size is probably not known beforehand and might also change with the complexity of the task. NEAT sidesteps all these insecurities by initiating training with minimal networks with no hidden nodes, and then increasing the size gradually until an increase in size does not result in progress anymore.

The solution to the two first problems lies in the genetic encoding of NEAT networks, and the inclusion of an *innovation number* or *historical marking* in every gene. The inspiration came from the natural process of synapsis were chromosomes are lined up according to genes encoding similar traits before crossover begins. The big realization that lead to development of NEAT was that when starting with minimal networks, mutations in different networks sharing the same historical origin would encode the same trait, and lining these up with each other would enable crossover. By following these historical markings, describing the time of creation of the gene, the competing conventions problem could be avoided.

The network has two different lists of genes. The first one specifies which nodes are available in the network and if they are of the input, output or hidden type. The other list has one gene for every connection between nodes. These connection genes describes which two nodes are joint by the connection, the weight, if the connection is active or disabled, and the innovation number telling when the connection was created. There are four types of possible mutations in these genomes. The first one is modification of a connection weight, either a small perturbation or a new random weight value. The others give structural changes in the network. A new connection can be created between two previously unconnected nodes. A new node can also be created in the middle of an existing connection, which would deactivate the old connection and replace it with two new connections joining the new node with the two old ones. Finally a disabled connection can get enabled again. Examples of the two first structural mutations are shown in Figure 2.2.

In crossover the genomes of two networks are lined up according to innovation numbers. If two genes have

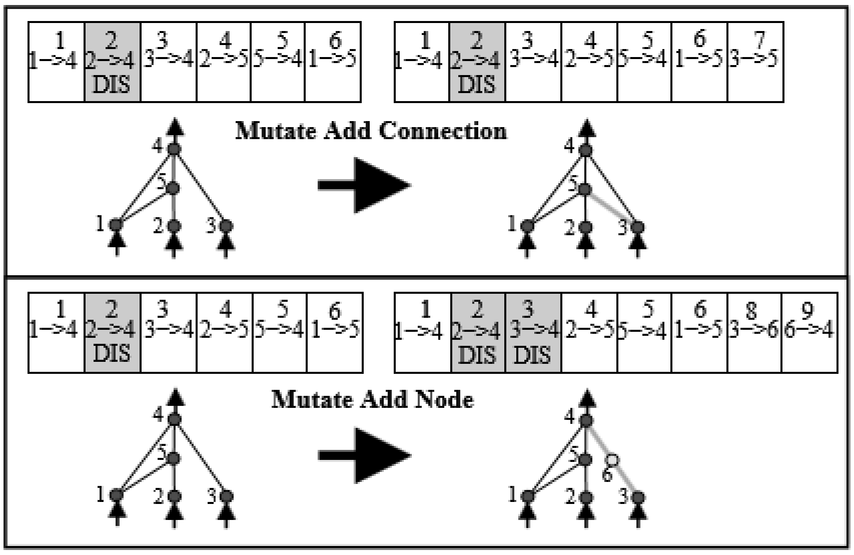


Figure 2.2: Two types of structural mutation in NEAT. The picture shows the connection genes and corresponding networks before and after the mutation. The genes show the innovation number at the top, then the in and out nodes, and finally if the connection is disabled. In the top part of the picture a new connection is formed between existing nodes 3 and 5. In the second part a new node is created, disabling the old connection between nodes 3 and 4 and replacing it with two new connections. Picture from [39].

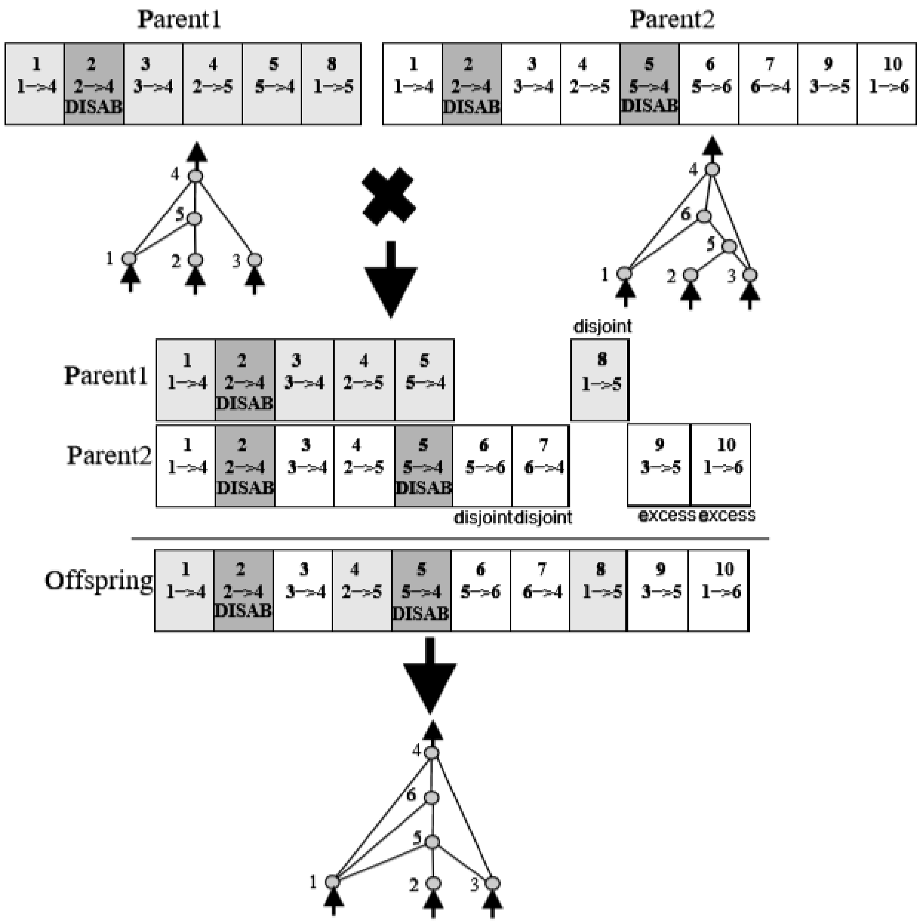


Figure 2.3: An example of crossover between two genomes in NEAT. These two parent genomes have matching genes up to innovation number 5. One of the genes from every matching pair is randomly chosen to be included in the offspring. If one gene in a matching pair is disabled there is a chance that the offspring gene will also be disabled. Excess and disjoint genes are normally included from the more fit parent, but in this example equal fitness was assumed so that the inclusion of these genes was randomized. Picture from [39].

the same number they are called *matching genes*, because they have the same historical origin and are thus guaranteed to represent the same structural change and be interchangeable. The other genes are either called *disjoint* or *excess*, depending on if their innovation numbers are within the same range as the other parent's genes or not, and represent structures that are not included in the other network. When the genomes are crossed, one gene from each matching pair is chosen at random and the rest of the genes are taken from the parent with the highest fitness. An example of crossover in NEAT is shown in Figure 2.3.

The historical markings also makes the comparisons needed for speciation possible. The compatibility distance between two networks can be calculated with the help of the number of disjoint and excess genes, and the matching gene weights. In each species one genome from the previous generation is set as a representative genome, and when a new network has a compatibility distance below a certain threshold from one of the representatives it gets assigned to that species. If no compatible species is found, a new one is created with the new genome as the representative. Every species will then be allowed to produce a number of offspring, depending on the average fitness of the species. There is also a procedure that can be activated to avoid stagnation. If the population fitness does not improve for a specified number of generations, only the two most promising species are allowed to reproduce.

3 Method

A literature study was performed of existing driver models and evolutionary machine learning techniques, seen in chapters 2.1 and 2.3. The purpose of the study was to decide which methods were suitable to implement and compare as representatives of the two conventions. As stated in the project description, in addition to the self-made model one was desired to be an established knowledge-based driver model and the other a neural network generated without prior knowledge. The models were then implemented, evaluated and compared in a scenario mimicking the FS trackdrive event. In this event the vehicle is following a narrow and winding lane marked only by cones on both sides. Since it is a racing competition, the objective is to keep a high speed in order to finish the race in as little time as possible. It is also important to stay on the track, since leaving the track is seen as a failure and hitting a cone results in a penalty. An example of a trackdrive track can be seen in Figure 3.1.

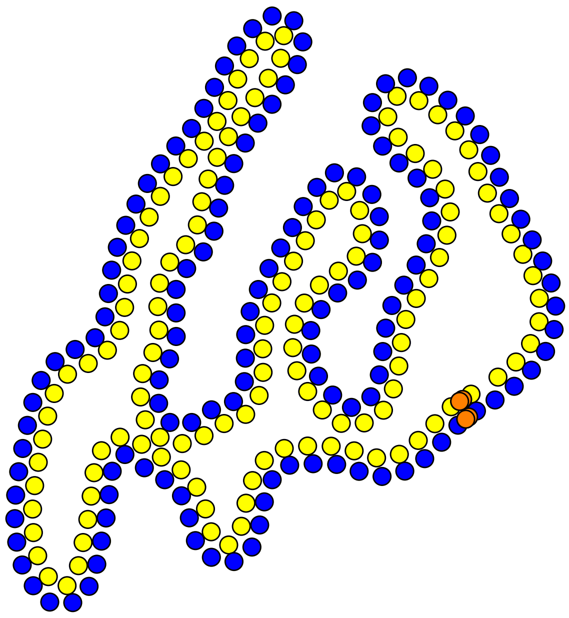


Figure 3.1: A trackdrive track seen from above. The left side is marked with blue cones and the right side with yellow cones. The start/finish line is marked with orange cones. Cone sizes are not to scale, the track is approximately 3 m wide and the side cone separation is approximately 5 m.

The implementations of both models were carried out in the C++ environment OpenDLV, a software platform for autonomous driving provided by Chalmers' vehicle laboratory Revere [42]. In OpenDLV, programs for autonomous vehicles are divided and deployed as a collection of microservices. This highly modular structure makes the program very flexible to modifications. For example, a fully functioning program for real world vehicles can easily be simulated by replacing the vehicle with a vehicle model, the localization module with a position integrator and the detection module with a virtual variant. The logic for deciding driving commands can thus be completely identical in both real world and simulated test cases. The opposite is also true, that the motion planning modules can be switched to other variants freely without affecting the rest of the simulation. This property is what was utilized in order to easily switch between the two knowledge-based models, and thus testing them in identical circumstances. The black-box model however required to have the cone detection, network and vehicle model in the same microservice in order to show the same behaviour in testing as in training. A picture of the simulation loop is shown in Figure 3.2.

An important aspect of making the comparison fair was to provide the driver models with identical inputs. In the scenarios that were chosen for the comparison in this thesis, the program was consistently aware of the positions of the five upcoming cones on each side of the track, equivalent to a preview of approximately 20 meters. Furthermore the program also knew the current vehicle state, more specifically the current longitudinal speed, lateral speed and rotational speed, i.e. yaw rate. The task of the driver models was to use this information to generate suitable steering and acceleration commands. Even though the implemented knowledge-based models

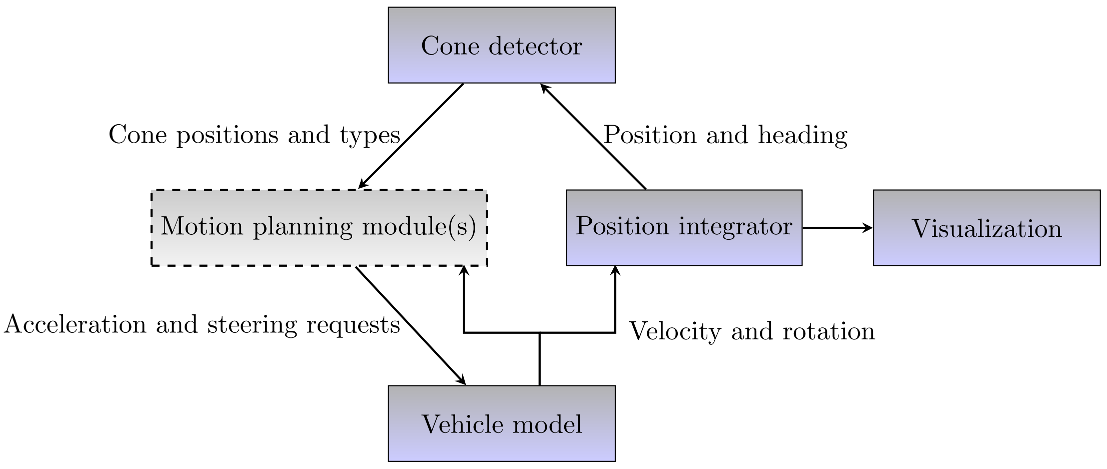


Figure 3.2: The architecture of simulation microservices and communication. A global cone map was availiable to both the cone detector and visualization microservice. A bicycle model was used as the vehicle model. The motion planning space was in the knowledge-based models filled with one microservice for path planning and another for steering and acceleration. In the black-box model the motion planning consisted of a trained ANN, and was included together with the cone detector and vehicle model in one microservice.

can handle a varying amount of detected cones, there are reasons to choose a fixed amount for the comparison. One is that the neural network in the black-box model needs a fixed amount of inputs to be reliably trained. Another reason is that this is how cones are planned to be sent to the driver model in the FS competition, as soon as the full track has been discovered and stored as a global map.

The self-made knowledge-based model, which will commonly be referred to as the *white-box model* or *aim point model* in this paper, is heavily inspired by the established aim point models introduced in 2.1 without being a direct implementation of any of them. It consists of a lateral control system using a single velocity dependent preview point, referred to as the aim point. The steering correction is based on the angular error between the aim point and the vehicle heading. The design is motivated by simplicity and few tunable parameters, and the use of preview allows it to be further developed into several more advanced models that are based on the same premise. The speed planning is represented by a speed profile, calculated from the maximum allowed lateral acceleration and the curvature of a path placed in the center of the lane. The speed profile is followed by estimating the latest time to brake compared to the time it would take to reach the considered position at current velocity. More detailed descriptions of the parts in this model are presented in sections 3.1, 3.2, and 3.3.

The chosen established knowledge-based model was the one by Sharp et al, commonly just referred to as the *Sharp model*. Since the model only handles steering, it replaced the aim point steering method from the white-box model while using the same path planner and velocity control method. The classic Sharp model has already been described in section 2.2. The implementation for comparison in this work was only slightly modified to compensate the lack of a global path. The used path planner was not guaranteed to give a path point directly to the side of the vehicle, and if this was the case the modified model chose the lateral offset to the position of the closest path point as the first lateral error. The effect of the modification is assumed to be negligible.

The neural network of the black-box model was generated with the neuroevolutionary technique NEAT. The main motivation for using a TWEANN is that it requires less input of prior knowledge from the developers. A more traditional neural network would require a decision for the network structure and size, and leaving these decisions to an evolutionary algorithm will increase the contrast between the black-box (no insight) and knowledge-based (full insight) models and make the comparison between the two approaches more clear. The reason for choosing NEAT specifically is that the relatively simple method has been shown to be able to perform well in complex problems. It is also well known and popular, so that online resources and examples can easily be found. An overview of NEAT was covered in section 2.4, and the process of evolving the network for this specific project is described in section 3.4.

3.1 Path planner

The first part of the white-box model involves placing a path in the lane. There are two purposes to making this path. One is that the aim point will be placed somewhere on the path, depending on the current speed of the vehicle. The other is that the curvature of the path will be analyzed in order to plan a suitable speed profile that will make the vehicle brake in preparation of a curve and speed up in straights. The path placement in the lane was discussed while developing the first version of the white-box model. One option was to find the optimal race line in the track, and make a driver model that can follow this path as closely as possible. It was however ultimately decided that the path should be placed in the center of the lane, for a couple of reasons. One is that safety and staying on the track is prioritized over speed, and purposefully guiding the vehicle closer to the edge of the track would introduce unnecessary risks. Another reason is that the implemented steering models might not necessarily be able to follow a planned path closely enough, and in that case the speed planner would have more use of the curvature of the center path that more closely represents the lane in general. Yet another reason is that the lane is very narrow, so that the available margin for lateral maneuvering is minimal and an optimal race line probably would not be able to deviate a significant distance from the center anyway.

The first things that happen in the path planning module are some checks on the detected cones, to make sure that they are able to represent a valid lane for path generation. These checks are more important in other test cases, when the vehicle can only detect cones within its field of view rather than always getting the five upcoming cones on each side, but they still serve as safety precautions in this simpler case as well. First it sees if the cones were received in the correct order, starting with the closest cones and ending with the farthest. This is how the cones are sent from the cone detector, but a double check is needed since messages between microservices cannot be guaranteed to be received in the same order as they are sent. Secondly the program investigates if the two sides match each other. This is done by first choosing to trust the side that is the longest, since it carries more information. It then makes sure that all the cones on the trusted side can see any cones from the other side within a certain distance threshold, and if not the other side needs to be supplemented. For example if the long side extends much farther than the short side, the program will insert guessed cones at suitable positions at the end of the short side. An example can be seen in Figure 3.3.

After the cones have been checked and possibly supplemented, it is time to generate a path in the center between the two sides. First the longest side is divided into smaller parts by placing *virtual points* at regular intervals between the cones. The next part is a process to create a partner point on the other side for every one of these points. Every point finds the closest cone from the other side. The segments from that cone to the previous and following cone are then searched for a point with a perpendicular line from that point to the virtual point in question. If such a point is found it is set as the partner point, and if not the partner point will be set at the position of the cone that was found. This search for perpendicularity is important to ensure that the path will be kept in the center even when the lane turns. When a partner point has been set for every virtual point, the center point for each of these pairs is stored. The final path points are then placed at small regular intervals along this collection of center points. A picture of the procedure is shown in Figure 3.4.

The path could be processed further than this, but was opted to stay in the form of a sequence of linearly interpolated points. The main reason is to keep the lane representation as simple and general as possible, and only refine the path when it is absolutely necessary for following driver model functions. Representing the path as for example a function or spline would on one hand provide a continuous reference which would be advantageous, but it would also introduce uncertainties in the function fitting process and require more computational power. Since both knowledge-based models were found to respond well to the point representation of the path, the advantage of having a continuous path was not deemed valuable enough to compensate for the disadvantages.

3.2 Aim point steering

The white-box model applies a single point preview steering model, here referred to as the the aim point model due to its single point characteristics. The method utilizes the basic principles of most traditional driver models where the approach is to look ahead a preview distance in front of the vehicle and decide an appropriate steering command based on the vehicle's deviation to the upcoming path. The aim point model could therefore

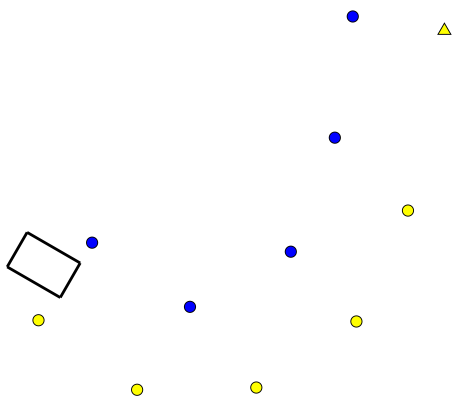


Figure 3.3: The car (box) has detected five cones (circles) on both sides of the lane. The left side extends much longer than the right side, so the program guesses an extra cone on the right side (triangle).

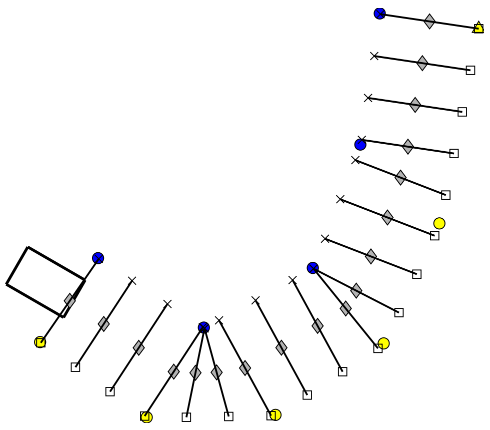


Figure 3.4: The path planning procedure. The right side is the longest and is thus divided with regularly placed virtual points (squares). Each virtual point gets a partner point on the other side (crosses). The center point (diamonds) of every pair is found and the path is placed between these center points.

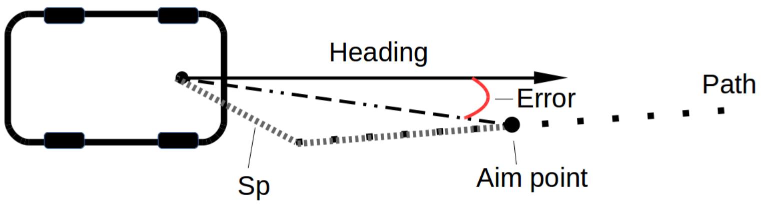


Figure 3.5: The aim point is placed on the path at a preview distance S_p calculated from the centre of the vehicle's front wheel axis. Steering corrections are based on the angular error between the vehicle heading and the aim point.

be motivated to be a proper representative for knowledge-based driver models. As input, the model receives information about the path from the path planner given in local two-dimensional Cartesian coordinates where the origin is placed at the center of the front wheel axis, the x axis is straight ahead of the vehicle and the y axis is to the left. A second input is given as the vehicle ground speed. The aim point is placed on the path in front of the vehicle at a preview distance S_p, as shown in Figure 3.5. When driving at high velocities, which occurs on straight road sections, it is advantageous to strive for smaller steering corrections to achieve a stable behaviour. While at lower velocities connected to cornering, it is important not to place the aim point too far ahead which would induce corner cutting. In order to achieve this type of adaptive behaviour the preview distance was made linearly dependent of the longitudinal velocity $\dot{x}$ using Equation (3.1).

$$S_p = \dot{x}T_p \tag{3.1}$$

The preview distance is calculated using T_p which is a set preview time. A great advantage with the aim point steering model is that it only contains one single tuning variable, which is the preview time. When the preview distance is decided, the xy-coordinates for the aim point is calculated by finding the path point of which the distance from the local origin is just surpassing the preview distance. Linear interpolation is used to cover the remaining error.

Two additional cases can be identified. If the preview distance is larger than the path length from vehicle, the aim point is placed on the last path point. If the preview distance is smaller than the distance between vehicle and the first path point, the aim point is placed on the first path point. The desired heading δ_{des} is then calculated in Equation (3.2) as the angle between the vehicle x-axis and the aim point.

$$\delta_{des} = \arctan 2(y_p, x_p) \tag{3.2}$$

Here, y_p, x_p is the placed aim point in local coordinates. The heading request is in turn limited to $-\delta_{max} \leq \delta_{req} \leq \delta_{max}$ due to vehicle specific mechanical restraints, as seen in Equation (3.3) where the heading request is limited by a maximum steering angle of ± 25 degrees.

$$\delta_{req} = \begin{cases} \min(\delta_{des}, 25\pi/180) & \text{if } \delta_{des} \geq 0 \\ \max(\delta_{des}, -25\pi/180) & \text{otherwise} \end{cases} \tag{3.3}$$

3.3 Velocity control

Here, velocity control refers to deciding a desired acceleration at each time point where a vehicle is aiming to drive as fast as possible around a narrow two-dimensional, high curvature track with all four wheels inside the road markings. As inputs, the velocity model receives information about:

- the path from the path planner given in local two-dimensional Cartesian coordinates where the origin is placed at the vehicle's center of gravity
- estimated road friction coefficient
- current heading error (i.e δ_{req})
- current ground speed
- allowed maximal acceleration, deceleration and velocity.

The first step is to create a speed profile where a maximum allowed velocity v_{max} is set at the path point coordinates. The velocities are determined by Equation (3.4) using the path curve radius R and a maximal lateral acceleration limit, $a_{y,max}$. To avoid infinity values on the straight road sections, the velocity is further limited by an upper bound, denoted by v_{limit}.

$$v_{max} = min(\sqrt{a_{y,max} \cdot R}, v_{limit}) \tag{3.4}$$

The maximal lateral acceleration, $a_{y,max}$, is calculated from the friction condition in Equation (3.5), where g and μ are the gravitational constant and estimated road friction respectively.

$$\sqrt{a_y^2 + a_x^2} \leq g\mu \tag{3.5}$$

The longitudinal acceleration, a_x, is assumed to be the maximal allowed for both brake and torque, to be safe, but could be seen as a tuning variable. The curvature is represented as the radius R of a circle that goes through three path points separated by a step size that is heuristically determined. The three path points make up a triangle with side lengths $A \geq B \geq C$, and the triangle area is calculated using Equation (3.6). The radius of the circle is finally determined by (3.7).

$$\triangle(A, B, C) = \frac{1}{4}\sqrt{(A + (B + C)) * (C - (A - B) * (C + (A - B)) * (A + B - C))} \tag{3.6}$$

$$R = \frac{ABC}{4 \triangle (A, B, C)} \tag{3.7}$$

This is done for as many path points that is possible with the given step size, and a reference velocity is set for all points that has a curvature estimate, using Equation (3.4). This method of curvature estimation is rather unconventional, but has proven to be competent together with the path planner. In relation to other methods, such as curve fitting which also was considered, it is computationally efficient and unaffected by the length of the path. When compared to a curve fitting method, the accuracy of the estimations were rather similar, however the circle estimator showed to be more consistent and forgiving in certain scenarios. It also has the advantage to be able to perform estimations on paths that changes sign in both x and y direction. Before moving to acceleration calculations, the speed profile is scaled down based on the quotient between the heading request and the wheel angle limit, $\lambda = \frac{|\delta_{req}|}{\delta_{max}}$ according to

$$v_{des} = v_{max}(1 - \lambda C)$$

where C is a tuning constant between 0 (no scaling) and 1 (full scaling, $v_{des} = 0$ when $|\delta_{req}| = \delta_{max}$). A $C = 0.5$ has been found sufficient.

To calculate the desired acceleration at time t, the basic concept was to find the latest time when the vehicle has to apply its brakes. The achievable future velocity when braking for a time t_b with constant deceleration $a_{x,maxdec}$ is denoted by $v(t + t_b)$. Equation (3.8) estimates the brake time to achieve the future desired velocity v_{des} while Equation (3.9) estimates the time t_v to reach the considered velocity at distance s with current speed $v(t)$.

$$t_b = \frac{v_{des} - v(t)}{a_{x,maxdec}} \tag{3.8}$$

$$t_v = \frac{s}{v(t)} \tag{3.9}$$

Since $a_{x,maxdec}$ is a negative value, t_b is positive when the current velocity is greater than v_{des}, which indicates that the vehicle is in need of deceleration to reach v_{des}. Three cases can be identified:

- $t_v - t_b = 0$, brake now to reach v_{des}.
- $t_v - t_b > 0$, braking is not yet necessary.
- $t_v - t_b < 0$, braking now will cause $v(t + t_b) > v_{des}$.

At each time t, the preview path is evaluated and the most critical preview point is identified, i.e. $\min(t_v - t_b)$. If the value is less than or equal to zero, maximum braking is requested. To not use exclusively maximal braking commands, another condition is included where if braking is needed within t_s, an appropriate deceleration is requested to reach $v_{des} = v(t + t_b + t_s)$, as well as a condition saying that if the deceleration can be achieved using only roll resistance then the requested acceleration is equal to zero. For $\min(t_v - t_b) > t_s$, maximal acceleration is requested. Before sending the acceleration request, the condition in (3.5) is checked again with

$$a_y = \frac{v_{current}^2}{L/tan(|\delta_{req}|)}$$

$$a_x = a_{x,req}$$

where L is the vehicle wheel base and $a_{x,req}$ the requested acceleration. If it is not fulfilled, $|a_{x,req}|$ is reduced using (3.5) such that the condition is met.

3.4 Evolutionary network generation

The networks were generated and optimized using the published code for the original version of NEAT [43]. The program requires a user defined experiment, defining the objective and dynamics of the task, and specified parameter values. The experiment is run for every organism in every generation to evaluate a fitness, and actions such as mutation, reproduction and speciation are all handled by the provided code. The experiment for this project was set as running the complete simulation loop. Provided a global position, the 10 nearest cones where sent as input to a network connected to a specific NEAT organism, together with the lateral, longitudinal and angular velocities. The network outputs, represented between -1 and 1, were mapped into steering and acceleration requests as $\delta_{req} = out_1 \cdot \delta_{max}$, $a_{x,req} = out_2 \cdot a_{x,max}$. These where then used by the vehicle model described in Section 3.5, which returned a kinematic state allowing the car to drive along a track by integrating new positions and create new inputs. The fitness of an organism was set as the average of the fitnesses of all the runs in the training tracks. There were in total 66 tracks available to the network. The training set consisted of 56 of these tracks, and the remaining 10 were used for testing. All the tracks were custom-made to ensure that the properties in terms of cone placement would be the same as the real event track, although with different track shapes. The number of unique track shapes was 20, but to enlarge the set tracks were reused by changing driving direction and/or scaling in size. The track shapes are shown in Appendix C.

The fitness was based on the competition event objective, which is to drive as fast as possible while staying in the lane. In the competition, leaving the track is seen as a *did-not-finish* (DNF), and hitting a cone will result in a penalty. The fitness function thus needed to be carefully designed in order to evolve a desired behaviour. Since the main priority was to find a safe behaviour that stayed on the track, both leaving the track and hitting a cone were set as failure conditions. The fitness was defined as the distance travelled along the track within a set time. This implicitly trained both speed and survivability at the same time, encouraging the car to go as far as possible before the allotted time limit was reached. Entering a failure state would end the attempt before the time was up, but the fitness would be set as the distance covered before failing so that even small improvements would be noticed and rewarded.

When experimenting with NEAT, there are a large number of parameter that can be altered to tweak the evolution, examples would be population size, mutation and mating probabilities, age significance, stolen babies etc. To optimize these is a difficult task which demands either genuine knowledge about the NEAT technique and its software, or extensive experimentation. This is considered to be outside the scope of this work. Therefore, the parameters recommended by NEAT's creator for the double pole balancing experiment are used as default setting for the evolution of the network. The parameters can be found in the original *C++* version of NEAT and in Appendix A.

3.5 Vehicle model

A vehicle model was used to transform the heading and acceleration requests into vehicle motion. The choice of model resulted in a conventional single-track vehicle model, seen in Figure 3.6, with simplicity as the strongest motivation. The model, also called the bicycle model, makes the simplification of modeling a four wheeled ground vehicle as two wheeled. Here, the vehicle is front steered, and modelled in two dimensions assuming no roll or pitch motions. The vehicle's motion used to integrate its global position was modeled as a kinematic state consisting of velocities in x and y directions, as well as the yaw rate.

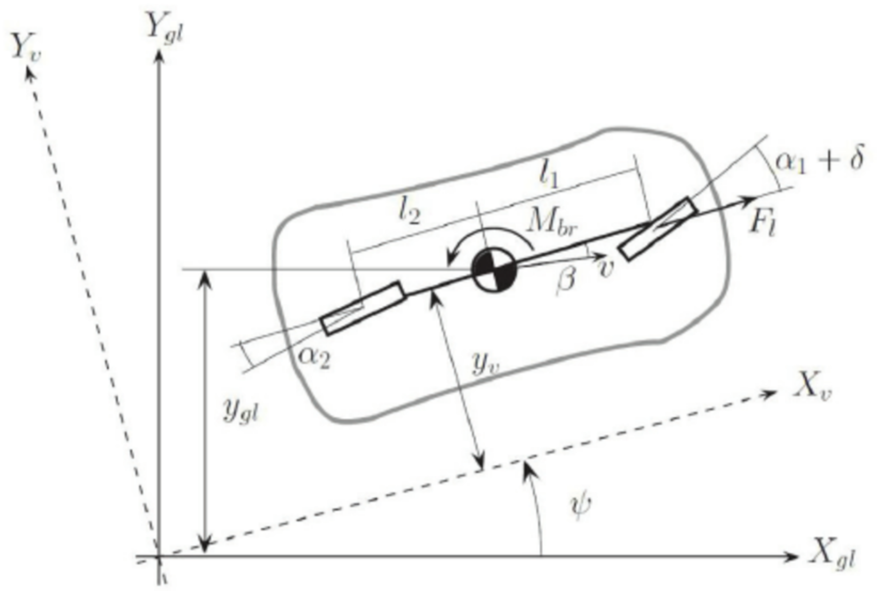


Figure 3.6: Single-track vehicle model [44]

$$\alpha_2 = -\arctan\left(\frac{v_y - l_2\dot{\psi}}{|v_x|}\right) \tag{3.10}$$

$$\alpha_1 = \delta - \arctan(\frac{v_y + l_1\dot{\psi}}{|v_x|}) \tag{3.11}$$

The physical vehicle parameters were estimations of the CFS17 race car and can be seen in Appendix B. The received heading request was realized by allowing a steering rate of at most $80°/s$. In the model, slip angles are estimated by equation (3.10) and (3.11), the rear and front ground forces are given as a function of the vehicle's mg and the distances l_2 and l_1 from Figure 3.6. The lateral forces on the tires $F_{1,y}$, $F_{2,y}$ are modeled using *Pacejka's magic formula* [45]. Finally, the v_x, v_y and yaw rate $\dot{\psi}$ of the current state are calculated by numerically integrating $\dot{v}_x$ (3.12), $\dot{v}_y$ (3.13) and $\ddot{\psi}$ (3.14) and adding them to the previous state. Additional modifications to the model were made by adding a constant roll resistance force F_r as well as limiting the longitudinal velocity according to $0 \leq v_x \leq 30$ m/s.

$$\dot{v}_x = a_{x,req} - \frac{F_{1,y}}{m}\sin(\delta) + \dot{\psi}v_y + F_r \tag{3.12}$$

$$\dot{v}_y = \frac{F_{1,y}\cos(\delta) + F_{2,y}}{m} - \dot{\psi}v_x \tag{3.13}$$

$$\ddot{\psi} = \frac{l_1 F_{1,y}\cos(\delta) - l_2 F_{2,y}}{I_z} \tag{3.14}$$

4 Results

To evaluate the three driver models, they were tested at ten different test tracks. Every driver model was allowed one try to finish one lap at each track and receive a score consisting of the time with added penalties. Naturally, a good model strives towards lowering its score as much as possible. According to Formula Student rules, hitting a cone results in a 2 second penalty added to the total time. An off-course occurs if, at some point, all four wheels are placed outside of the track and results in a DNF. The black-box model was trained on the tracks in the training set, which did not include the test tracks. The two knowledge-based driver models were first tuned on a couple of the training set maps, prioritizing safety, before being tested. The results are displayed in Table 4.1. The one called TrackFSG is a bit special, because it is a reconstruction of the 2017's edition of Formula Student Germany's endurance track for non-driverless vehicles. The track was generated using GPS data from the CFS17 vehicle. Comparing the models on this specific track is good for evaluting their performance on a track with the same properties as they would encounter in a real competition. Because of an off-course from the black-box we will however instead use TrackFSGR for this comparison, which is the same track with all the same properties but in the reverse direction. A closer look on the results from this track is shown in Table 4.2. Examples of the positioning of the three models from this track can also be seen in Figure 4.1 and 4.2.

Table 4.1: The results from the ten test tracks. The 'b' and 's' in some track names denotes that it is a bigger or smaller version of the normal track, and R means reverse. The result of a run is shown as *lap time* and *cone hits*. At the bottom the sum of the FS trackdrive scores from these runs is shown as a measurement of the driver model's performance. A low score is better than a high score.

	Aim point	Sharp	Black-box
Track19	62.35 | 0	64.80 | 0	DNF
Track19R	64.45 | 0	65.20 | 0	DNF
Track20	43.85 | 0	45.05 | 0	90.10 | 5
Track20R	43.55 | 0	44.65 | 0	DNF
Track20b	57.70 | 0	59.35 | 0	138.2 | 1
Track20bR	57.05 | 0	58.85 | 0	146.3 | 4
TrackFSGs	81.00 | 0	83.75 | 5	DNF
TrackFSGsR	83.35 | 7	83.45 | 8	DNF
TrackFSG	102.7 | 0	105.0 | 0	DNF
TrackFSGR	102.5 | 0	105.6 | 1	245.3 | 6
Total score	712.5	743.7	-

Table 4.2: The results from one lap in the reverse FSG 2017 track, with models tuned and trained only on the training set.

Model	Lap time (s)	Avg speed (km/h)	Left cones hit	Right cones hit	Event score (s)
White-box	102.5	37.61	0	0	102.5
Sharp	105.6	36.50	0	1	107.6
Black-box	245.3	15.74	6	0	257.3

To not only see how general and flexible the models were, but also to see more of how good they could potentially get, an additional test was made in the non-reversed FSG track. The models were now tuned and trained directly on the FSG track. The results were not aimed to become completely optimized, but at least get closer to their full potential to be able to see if any conclusions could be made regarding possible skill ceilings of the models. The results from these attempts are shown in Table 4.3.

Table 4.3: FSG lap results, with models tuned and trained specifically for the track.

Model	Lap time (s)	Avg speed (km/h)	Left cones hit	Right cones hit	Event score (s)
White-box	99.85	38.62	0	0	99.85
Sharp	101.9	37.85	0	0	101.9
Black-box	138.9	27.80	10	1	160.9

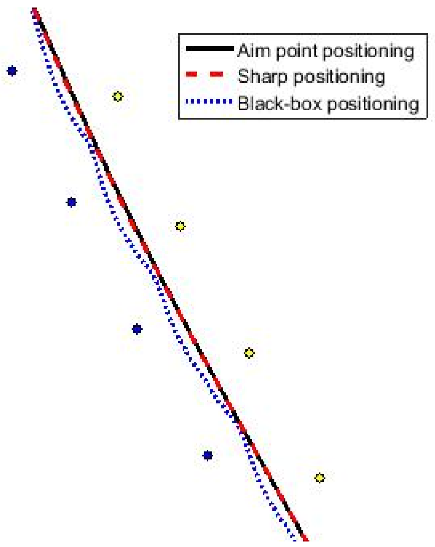


Figure 4.1: Example of positioning in a straight. Aim point and Sharp are nearly identical and centered. Black-box moves forward in a wavy pattern.

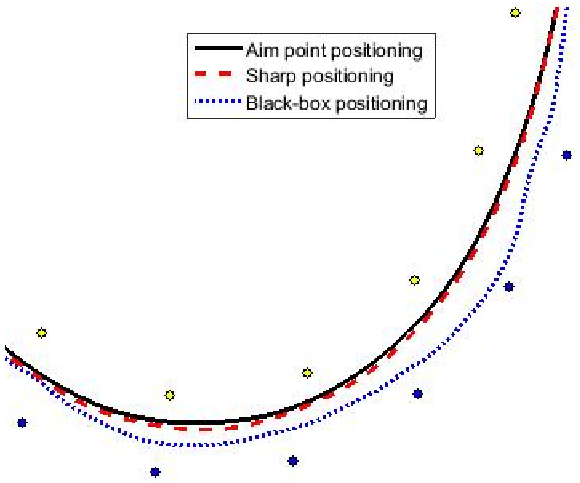


Figure 4.2: Example of positioning in a turn. Aim point and Sharp are nearly identical and slightly corner cutting. Black-box has chosen a very different trajectory.

To evaluate the second research question, and compare how the models respond to changes in the vehicle platform, the vehicle model parameters where changed such that the vehicle would resemble a large and heavy Volvo XC90 instead of the lightweight Formula Student race car. Weight, length and inertia were altered while maximum and minimum values for steering and acceleration where left unchanged. Since this created a vehicle that could impossibly managed to complete the FSG track due to its large steering radius, the models where tested on the easier Track20b. All of the models drove off track on the first try, as seen in Table 4.4. However, the aim point model only needed a change in the velocity control scheme to finish the lap, while the Sharp module got the same change and an additional tuning in steering to achieve a decent result. Both models hit the same three cones when exiting the sharpest corner of the track. The black-box also drove off the track, and could not be tuned without also retraining.

Table 4.4: The results on Track20b with changed vehicle parameters. Both with the tuning used for the old parameters, and retuned to fit the new parameters.

Model	Lap time (s)	Avg speed (km/h)	Left cones hit	Right cones hit	Event score (s)
White-box	-	-	-	-	DNF
White-box retuned	98.6	22.37	0	3	104.6
Sharp	-	-	-	-	DNF
Sharp retuned	93.45	23.61	0	3	99.45
Black-box	-	-	-	-	DNF

The black-box model used for this experiment was the model trained specifically on the FSG track. Interestingly, with 1 cone hit and average speed on 20.73 km/h it managed to finish a lap on Track20b in 106.7s. This is 31.5s faster than the model that was trained on several maps to achieve a robust behaviour in most situations, despite only training on one single track that was not even the testing track. Similar results were found when testing the FSG-trained model on other tracks, it consistently performed better than the other model.

4.1 Knowledge-based driver model results

Both knowledge-based models performed well on the ten test tracks. The aim point model managed to drive through almost all the tracks without hitting any cones, while the Sharp model hit a few more. The two models are however very close to each other regarding performance, both in speed and score. The white-box model's lap in the reverse FSG track was made in 102.5 seconds and Sharp finished in 105.6 seconds, which can be compared to the fastest lap times made by drivers in FSG2017 which were set close to 70 seconds on a

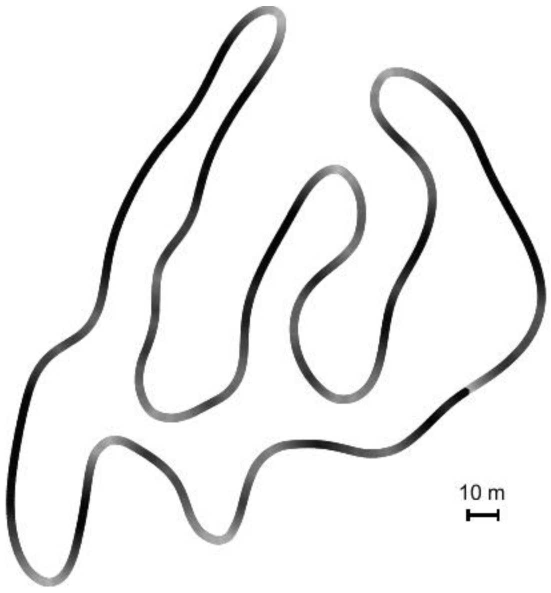


Figure 4.3: Visualization of the speed of the white-box model during one lap in the track. The speed is represented in a gray scale where the darkest black corresponds to the top speed of 15.19 m/s, and pure white corresponds to 0 m/s. Since the Sharp model uses the same velocity control scheme, the figure is representative for both knowledge-based models.

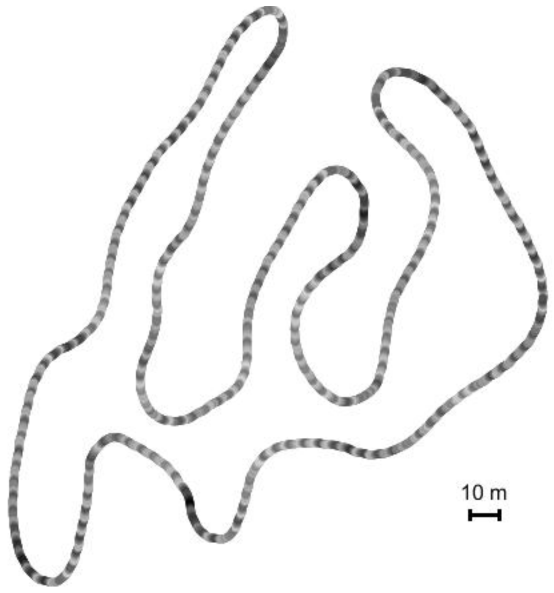


Figure 4.4: Visualization of the speed of the black-box model during one lap in the track. The darkest black corresponds to the top speed of 10.34 m/s, and pure white corresponds to 0 m/s.

wider track with the same layout[1]. The aim point model did not steer into any cones during its lap, while the Sharp model hit one cone. The vehicle longitudinal velocity throughout the lap was nearly identical for both knowledge-based models, since they were using the same velocity control method, and is illustrated in Figure 4.3. The start position can be spotted to the lower right where a dark higher velocity abruptly overwrites the white of the start position. Throughout the lap, the driver model manages to lower the speed into the tight corners and holds the positive acceleration until the wheels have been straightened. One can see a clear connection: the higher curvature the road has, the lower is the velocity, and vice versa.

4.2 Neuroevolutionary driver model results

The winning artificial neural network trained on several maps resulted after 4286 generations. The network consists of 23 input neurons, 1 bias, 554 hidden neurons and 2 output neurons. The neurons are in turn connected by 1871 weighted connections. When evaluated on 56 different maps during a maximum drive time limit of 180s, the network managed to achieve an average distance driven of 192.53 meters. The evolution is illustrated in Figure 4.5. The neural network trained specifically for the FSG track received its maximum fitness of 263.21 meters after 11801 generations, resulting in a larger network with 1427 hidden neurons and 4839 weighted connections. The more lightweight training allowed for faster evaluations of each generation.

During the testing the first network finished four of the ten tracks with a couple of penalties due to cone hits, and an DNF on the rest of the tracks due to Off-courses. The reverse FSG lap was finished in 245.3 seconds with 6 cone hits. In Figure 4.4, the vehicle longitudinal velocity throughout the reverse FSG lap is illustrated in a black and white scale. The car was driving slowly and carefully in the lane. There were no long accelerations or brake actions, instead the car accelerated and braked with quick bursts throughout the whole lap. It is however clearly shown that the speed was generally higher just before entering a curve. The second network, trained only on one map, showed an increase in performance regarding velocity control. This can be seen in Figure 4.6, which is somewhat beginning to look more similar to Figure 4.3. As a final remark, both of the networks developed a steering technique that involved consistently altering between maximum an minimum steering angles, even on straight road segments where the frequency of the altering simply increased.

[1]`https://www.formulastudent.de/fsg/results/2017/`

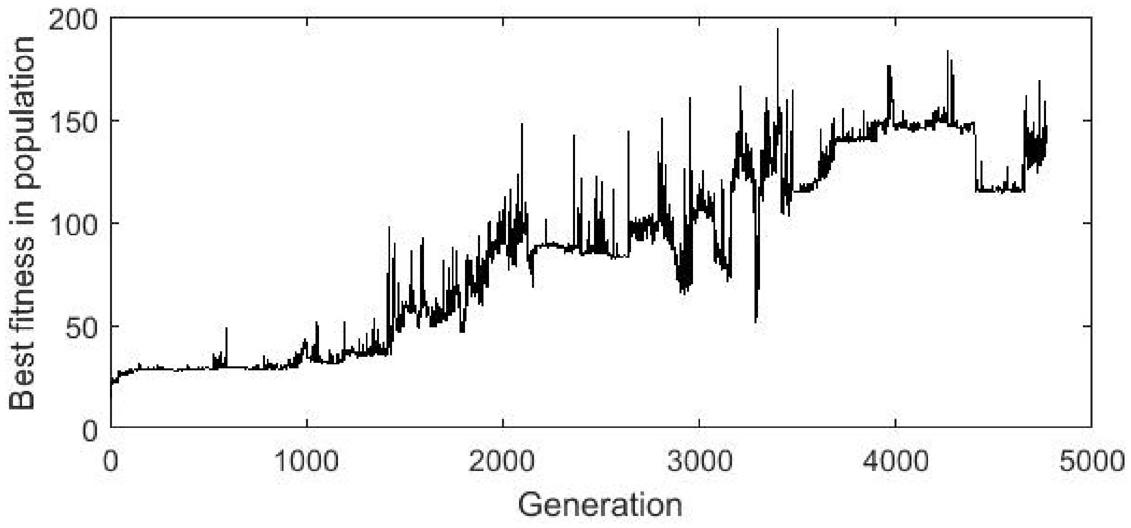


Figure 4.5: Population highest fitness for each generation during NEAT evolution when using a training set of 56 maps.

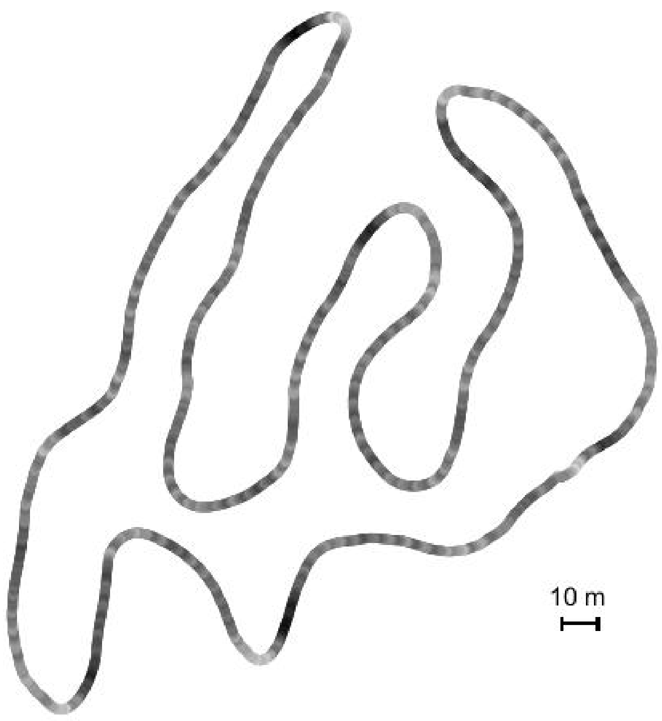


Figure 4.6: Visualization of the speed of the black-box model trained on the FSG track. The darkest black corresponds to the top speed of 15.30 m/s, and pure white corresponds to 0 m/s.

5 Discussion

The results were somewhat expected, in the sense that the two knowledge-based models would perform better than the black-box. We did not expect the black-box model to receive a DNF in so many of the test tracks, but the performance looks much worse than it actually is. When only looking at the scores the black-box might look worthless, but it actually was very skilled at avoiding the cones in general. The evidence can clearly be seen in the results of the tracks it managed to finish. In Track FSGR for example, 6 cone hits might look bad compared to the 0 and 1 by the other models. But then it is important to remember that the track consists of a total of 433 cones, and the black-box thus avoided 98.61 % of all the cones it encountered on the way and stayed on track the entire time. The locations where it unfortunately drove off track were also to a large extent where the other models struggled a bit, either with cone hits or close calls.

While the black-box steering model succeeded quite well in its most important task of staying on track and avoiding cones, it can however not be considered a valid steering model for racing. The steering request was almost always to the maximum angle to either the left or right. The network had during training discovered a steering technique that quickly alternated the aim between left and right. When going straight the aim alternated equally to both sides in an even frequency. When the car was turning the aim continued to alternate but stayed slightly longer to the correct side every time, except in the very sharp turns where the aim was held to the correct side. This steering method technically works in the sense that the resulting car movements are in the correct directions, but it also has a negative effect on the lap time because of the slight zig-zag pattern it always performs throughout the lap. And while it might be technically functional in a simulation, there would be a high risk of damaging the mechanical parts in a car if it would be implemented in a real world system. A contributing reason to why the model developed such a behaviour was possibly the less advanced vehicle model. Other than damaging components, in a real world scenario the vehicle system would itself have higher inertia, that is, being less reactive and follow the requests of the network more slowly. The constantly interchanging steering commands could as well make the system largely unstable, especially at higher speeds.

Regarding the black-box velocity control, the evolved solution was also surprising. The car drove with quick bursts of acceleration and braking in a seemingly nearly constant frequency. It is unclear why this behaviour evolved. Especially in parts of the tracks where it looked obvious that a longer acceleration or at least a constant speed would be better. A possible theory is that the model had a desire to accelerate, however, the large steering angles often created a situation where the vehicle then would accelerate toward a cone, forcing immediate braking. Another is, since the speed and steering were decided at the same time in the same network, they are likely coupled with each other. It is possible that the stuttering velocity control was caused by the quick jumps in steering, or the other way around. It might be that the model recognized that, just as for the knowledge-based models, a large steering correction should be connected with a low velocity. It is also possible that they were not at all dependent on each other and that this specific combination of steering and speed control reached some kind of local optimum. It is of course impossible to tell because of the black-box characteristics.

The fact that the aim point model would do well and be able to finish most laps without cone hits was not surprising, after all the times it has been tried and refined during the development process. It has been developed with the specific goal of succeeding in the competition, and it is easy to tune, so it was pretty certain that a car with slightly careful tuning values would be able to perform well even in unknown tracks. It was more surprising that Sharp got so similar results to the aim point model, even down to the positioning in the lane. Since they are using the same path planner, and Sharp is known for its path following ability, that indicates that also the self made aim point model is good at following the given path and on par with established steering models. The velocity control also worked as intended, quickly speeding up in straights and choosing suitable speeds for all turns, allowing the steering to work at its best when the risk of spinning out is the highest. And the fact that it worked equally well for two different steering models is a good sign of its flexibility. Worth mentioning is that the tuning of maximum velocity for the speed profile proved to be highly dependent on the amount of input cones to the system, i.e. how far ahead the driver model can see. When allowing the model to receive twice the distance ahead, the velocity on straights could easily be increased by more than 5 m/s. The Sharp model is recognized as an expert path follower, when tuned correctly. Due to the many parameters however, it is a time demanding task and it is possible that it did not show its full potential in this implementation. Moreover, the slight modifications made to the model are not expected to, but might have affected its performance, something that was not investigated thoroughly. The model showed less smooth steering corrections between sample times than the aim point model. This could be due to above mentioned reasons, or to the fact that the path planner was not designed to provide a path with upper bounded curvature.

Having a smoother path representation than linearly interpolated points could perhaps improve the results. The change in the aim point steering is assumed to be minimal, because with a discrete time step the path would be percieved as discrete points anyways, but it could make a difference in the Sharp model that is dependent on the accuracy of relative positions of several path points in any given time frame. If such a change would have a noticeable effect on the performance is however unclear. The largest difference might have happened in velocity control, which could potentially be designed to make use of a more accurate curvature estimation. While on the topic of paths, it would have been interesting to see if the black-box model would become better if the input of cone positions was replaced with a path input, either the current point representation or a smoother one. It feels like it might be easier to learn how to follow a path rather than how to go fast while avoiding suddenly appearing cones. There are however several issues with trying this. For one it might be hard to make local paths consistent enough to be able to act as reliable training data, and in this case it would for example be hard to guarantee a certain path length. Secondly the model would be biased on where the designers think that a good path should be placed, which might restrict the performance from becoming as good as it can get. Lastly, this would give some insight in how the model decides its outputs, and while it might get better results it would not be a no insight model and therefore not in line with the purpose of exploring a black-box model in this thesis.

5.1 Advantages of knowledge-based models

We can see some advantages to controlling the car with a knowledge-based model. The first, and perhaps most important, is that we have full control over the safety precautions in the system. We can decide what situations should be regarded as critical, and what the appropriate action should be. If the situation is temporary the solution might be to ignore it to see if the danger is false or to slow down while waiting for a clearance signal. If the situation is serious an immediate emergency brake or quick turn might be warranted. Of course it might be difficult for a system designer to predict all the special cases, but it is possible when the task is not too complex. With a black-box model one would have to rely on having all the cases present in the training data and also that the program has learned appropriate actions to these cases by itself. One alternative could be to override the network output in special cases, but that would not only defeat the purpose of having a black-box model. It could also introduce problems and unexpected behaviour if the overriding was not present in training, or if the model is dependent on recurrent data.

It is also much easier to modify existing behaviour, not only in critical situations but also when the driving is proceeding as expected. If the car is driving much slower than what is desired, overall driving aggressiveness can be increased. If the car has a not quite optimal curvature to speed conversion or needs to brake or turn harder or earlier, there are tuning parameters for all of those. If the tuning parameters are not too many, it is not too hard for a human to slightly modify them until a good combination has been found. With a black-box model the only alternatives would be to either retrain with the same settings, retrain with new settings, or even altering the fitness function either greatly or slightly. All of these alternatives are likely to take a long time, and none are guaranteed to improve the results. In fact, even evolving two networks simultaneously on the exact same premise, the final results may differ largely. A well known problem in using black-box models as aid for autonomous driving is validation. It is very difficult to determine if a system is safe, well performing or robust enough in all situations. Since one cannot foresee its actions, testing is the only way to discover the limits of the system, but the conclusions are only as good as the amount of scenarios that are evaluated.

In the development process the advantage is similar to the one just discussed. In the search for a design that is able to in principle solve the basic task, at least on a somewhat satisfactory level, the full insight of knowledge-based models often made it possible to find and resolve different causes to problems separately, and gradually working towards a robust design. No major tuning is required until the design phase is over and the most basic level of the problem has been solved. In this case that could be considered to just staying in the lane by keeping the speed within reasonable limits and steering with the right principal direction in curves. To reach this point with a black-box model some uncertainties are introduced before training can even begin such as setting training parameters, setting a fully covering evaluation function for solving the task, and relying on that the training data has both the necessary information to learn the task at all and the necessary diversity to find a general solution. There are also no intermediate steps, either a good enough solution has been found or more training is needed.

This brings the discussion to real life development and implementation. A knowledge-based model has the advantage that it can be created and tested in simulation as proof of concept. Then when the software is deployed to the real vehicle, proper tuning is performed to achieve desired behaviour. The case is different for black-box driver models. When simulating the black-box model, it had to be set up as similar to the training scenario as the OpenDLV simulation environment allowed. This means that the network activation, vehicle model and cone detector all needed to exist as functions within the same data triggered micro service. The only difference lied in the position integrator, which was a time triggered stand-alone module that controlled the frequency of the system in 20 Hz. Having most of the functions in the same micro service resulted in a loss of the modularity that the knowledge-based models had, but was required to actually get the same behaviour in testing as the one evolved through training. Though, an attempt was made to separate the module containing the network into a set up similar to the one used by the knowledge-based models, where driver model, vehicle model, cone detector and position integrator all exists in their own micro services. With the short delays, in the order of milliseconds at worst, induced by sending messages, the performance of the black-box model dropped significantly. The conclusion is that when training a black-box driver model, it is important that training is performed in a highly similar setting to where it is meant to be used. This makes training in simulation a nearly impossible task when the software is to be deployed in a real vehicle. On the other hand, to perform the training in the setting where it is to be used, i.e. in the real vehicle, might be close to as challenging, especially if approaching the task as a reinforcement learning problem, as was done with NEAT. There is a high probability that the hardware would be damaged in the process, and even so, if training is performed outside simulation it would take a tremendous amount of time. As is the case with all machine learning techniques, the performance of the system is highly correlated to the amount of data that is used for training. Here, the behaviour cloning technique used in [7] is a promising approach as it has the ability to collect data and train the driving behaviour on any road without risking neither the environment nor the vehicle platform. It also avoids the *restart-retry* aspect of reinforcement learning.

As an additional remark, the knowledge-based models we have used are designed to be largely independent of the specific vehicle model used. The decision making for steering and acceleration would not need to change with a different vehicle, although some new tuning would probably be required in most cases. That is radically different from our black-box model, which is trained based on performance. This means that the training algorithm continuously needs to compare vehicle responses from different network outputs. A vehicle change would therefore, again, require a complete retraining of the network with new settings. It is however worth to mention that while this is the case for the models we have tested, it might not be the same for all knowledge-based or black-box driver models.

5.2 Advantages of black-box models

Using a black-box model also has its advantages. First of all it can be very quick to get started. While knowledge-based driver models are transparent, the underlying mathematics are often complex and could be demanding extensive knowledge in system control in order to achieve a stable and well behaving system. It might also be that existing models are not completely suited out of the box for the specific task in mind, and could either not be usable at all or require significant modifications in the function logic. For example, to try the Sharp model in this experiment we first needed a path to follow. Since the very specific properties of representing a lane with sparse cones is not very common there is no definitive and ready made way to represent the lane with a path, so we had to design and implement that by ourselves. Then the path representation we had available was not always completely suited for the traditional Sharp model, so that the logic also needed some small changes. Instead of modifying logic or making things from scratch, it is very convenient to pick a machine learning technique. Most algorithms are designed to be able to solve any problem, at least to some extent, as long as enough training data is available and the evaluation of performance can be clearly defined. In this project NEAT was used, which has a lot of downloadable resources available, but there are also numerous other machine learning techniques that should be able to find a solution. We believe that letting a finished algorithm start optimizing on enough data would in most cases be much quicker than creating a driver model from nothing. At least to start getting somewhat functioning results, even though tuning it to perfection might be more challenging.

Another possible advantage could be computation. A well structured network has the potential to represent a solution in a very compact way. While training a network requires a lot of computation power, the finished product requires a lot less to just run through. Especially for tasks that do no need particularly deep networks, and also with techniques such as NEAT that searches for minimal solutions. Running inputs through the finished network might be less demanding than using a knowledge-based model that always needs to do a lot of processing just to interpret the data in every time step, into a representation that must be comprehensible by a human. The difference in required computational power is something we would have liked to test in this project, but were unable to because of time constraints.

Maybe most importantly, the advantages of knowledge-based models quickly fade when the complexity of the task increases. While the knowledge-based models were ultimately preferred for the task in this project, it was heavily simplified compared to real traffic situations where driver models will most commonly be used. The scope of the FSG trackdrive task could easily be grasped by a human, and all the necessary steps to solve it could therefore be outlined in a self made model within a reasonable time. Neural networks, and black-box models in general, really show their true worth when a problem is too complex for a human. With more difficult and varying roads and environments there is no doubt that solving the entire task with a self made model would be too much of a chore for a human. While a human can drive a car and react with common sense in unexpected situations, transferring that knowledge and instinct to all the special cases in a knowledge-based model would require complete knowledge of the human brain. But when training a network it will optimize when and how to react to different input combinations, and in an unexpected situation it can react similarly to as it would in the most similar situation it knows about. It might not be a perfect reaction, but it has evolved something that could be referred to as instinct. It also might not be a perfect substitution to defining all special cases, but it is necessary when the amount of possible special cases is too large to grasp.

6 Conclusion

In this paper, an exploration of driver model conventions has been presented. On one side stands the more traditional knowledge-based algorithms, allowing for transparency and flexibility, and on the other side the neural networks with black-box characteristics and potential to achieve a behavioural complexity that could not be designed by a human. To evaluate both conventions, three different driver models were implemented. The first was a single preview point steering model, referred to as the aim point model, designed by the authors, the second was based on the well known model by Sharp, Casanova and Symonds, and the third was an artificial neural network created using the neuroevolutionary technique NEAT. The first two have been referred to as knowledge-based models and are also using a discrete path generated by a self designed path planner together with a speed profile to solve the task of requesting appropriate heading and acceleration commands based on a fixed number of cones marking out a drivable road surface.

Comparing the two conventions, the knowledge-based models proved to be more competent on the simulated test tracks. They both were clear winners in terms of speed and lane following abilities compared to the black-box model. The overall results were correlated to the ease of tuning, where the aim point model with preview time as the only tuning variable was the overall winner. The Sharp model performed nearly as well but was more difficult to tune well. With one setting it cut sharp corners too much and with another it took a larger turn than necessary. This was also dependent on the path planner and the velocity control, an interplay which was easier to create together with the aim point model. The aim point model proved to be very smooth and consistent with regard to steering corrections, while the Sharp model could make larger correction changes between time steps. The velocity control used on both knowledge-based models was very efficient, and is leaving room for shorter lap times if the possible viewing distance is increased.

The black-box, which was untunable, was lacking in terms of both steering and velocity. The model had developed logic for steering that consistently altered from maximum to minimum, even on straight road segments. It is difficult to draw conclusions about the velocity control, since it is possible that it was highly affected by the bad steering decisions. The knowledge-based models were overall considered more flexible to environmental changes, as they performed better in unknown tracks, and vehicular changes, as only some retuning was required when changing the vehicle parameters.

Overall, the presented task was easily comprehensible for humans. Both the transparent knowledge-based models could therefore fulfill the requirements and be tuned to improve the results. The black-box model did not perform remotely as well as the other models, but there are many reasons that could have affected the network which are not easily discovered and rectified due to the black-box characteristics. One would be that the simplistic vehicle model might have been too unrealistic. For humans it is easy to presume basic knowledge about vehicles and their physics that are unconsciously built in to a knowledge-based driver model, in the network evolution however there is no such knowledge in the system, which allows solutions that for us are obviously not desirable but can result in an evolution that takes an unwanted direction. The evolution of the networks in this thesis was performed over night, and tested the day after. It is possible that better results could have been achieved if training was allowed for a longer time. However, with time, the sizes of the networks increase rapidly, demanding more and more processing power to evaluate and continuously write results to file. Also, it is not certain that the training was set up in an optimal way. Several versions were tested, and it was discovered that training on only one map gave better results than to use a larger training set. It would be interesting to dig deeper into the evolution phase, and to aim for reaching the full potential of this NEAT experiment. This could be done by altering parameter settings for the training, try different fitness functions, training sets and validation methods. However, since it did not fit within the scope of this work, to properly investigate the best evolution set up is left for further development.

Nevertheless, the black-box model managed to finish several tracks, which is still impressive considering that it was evolved from only 24 input neurons and two output neurons, with loose instructions telling it to go far and stay on track without hitting cones. It is still difficult to compete with the inherited knowledge of the human brain when designing driver models. Though, the potential of ANN is higher when it comes to more complex and unexpected scenarios that becomes too difficult to model mathematically. Neuroevolution as such could allow for a low threshold to develop advanced neural networks with only basic knowledge about the task that is to be performed, and could therefore grow to become a powerful tool in numerous fields in the future.

References

[1] L. Caltagirone *et al.*, “LIDAR-based driving path generation using fully convolutional neural networks”, *arXiv preprint arXiv:1703.08987 v2* 2017, 2017.

[2] O. Benderius, *Modelling driver steering and neuromuscular behaviour.* Chalmers University of Technology, 2014.

[3] R. Sharp, D. Casanova, and P. Symonds, “A mathematical model for driver steering control, with design, tuning and performance results”, *Vehicle System Dynamics* **33**, no. 5 2000, 289–326, 2000.

[4] D. D. Salvucci and R. Gray, “A two-point visual control model of steering”, *Perception* **33**, no. 10 2004, 1233–1248, 2004.

[5] E. D. Dickmanns *et al.*, “The seeing passenger car’VaMoRs-P’”, *Intelligent Vehicles’ 94 Symposium, Proceedings of the*, IEEE, 1994, pp. 68–73.

[6] D. A. Pomerleau, “Alvinn: An autonomous land vehicle in a neural network”, *Advances in neural information processing systems*, 1989, pp. 305–313.

[7] M. Bojarski *et al.*, “End to end learning for self-driving cars”, *arXiv preprint arXiv:1604.07316* 2016, 2016.

[8] M. Bojarski *et al.* (Aug. 2016). End-to-end deep learning for self-driving cars, [Online]. Available: `https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/` (visited on 12/15/2017).

[9] J. Michels, A. Saxena, and A. Y. Ng, “High speed obstacle avoidance using monocular vision and reinforcement learning”, *Proceedings of the 22nd international conference on Machine learning*, ACM, 2005, pp. 593–600.

[10] K. O. Stanley. (Jul. 2017). Neuroevolution: A different kind of deep learning, [Online]. Available: `https://www.oreilly.com/ideas/neuroevolution-a-different-kind-of-deep-learning` (visited on 02/10/2018).

[11] F. P. Such *et al.*, “Deep Neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning”, *arXiv preprint arXiv:1712.06567* 2017, 2017.

[12] A. Tustin, “The nature of the operator’s response in manual control, and its implications for controller design”, *Journal of the Institution of Electrical Engineers-Part IIA: Automatic Regulators and Servo Mechanisms* **94**, no. 2 1947, 190–206, 1947.

[13] M. Kondo and A. Ajimine, “Driver’s sight point and dynamics of the driver-vehicle-system related to it”, SAE Technical Paper, Tech. Rep., 1968.

[14] J. Baxter and J. Y. Harrison, “A nonlinear model describing driver behavior on straight roads”, *Human Factors* **21**, no. 1 1979, 87–97, 1979.

[15] M. Land and J. Horwood, “Which parts of the road guide steering?”, *Nature* **377**, no. 6547 1995, 339–340, 1995.

[16] C. C. MacAdam, “Application of an optimal preview control for simulation of closed-loop automobile driving”, *IEEE Transactions on systems, man, and cybernetics* **11**, no. 6 1981, 393–399, 1981.

[17] O. Benderius, *Modelling driver steering and neuromuscular behaviour.* Chalmers University of Technology, 2014.

[18] T. Gordon and N. Magnuski, “Modeling normal driving as a collision avoidance process”, *Proceedings of 8th International Symposium on Advanced Vehicle Control* 2006, 2006.

[19] O. Benderius, “Driver modeling: Data collection, model analysis, and optimization” 2012, 2012.

[20] P. LIMA, “Predictive control for autonomous driving”, PhD thesis, PhD thesis, KTH, 2016. Unpublished thesis, 2016.

[21] P. Falcone, “Nonlinear model predictive control for autonomous vehicles”, PhD thesis, Università del Sannio, 2007.

[22] M. A. Abbas, “Non-linear model predictive control for autonomous vehicles”, PhD thesis, UOIT, 2011.

[23] R. Lenain *et al.*, “Model predictive control for vehicle guidance in presence of sliding: Application to farm vehicles path tracking”, *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, IEEE, 2005, pp. 885–890.

[24] T. Fraichard and A. Scheuer, “From Reeds and Shepp’s to continuous-curvature paths”, *IEEE Transactions on Robotics* **20**, no. 6 2004, 1025–1035, 2004.

[25] L. E. Dubins, “On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents”, *American Journal of mathematics* **79**, no. 3 1957, 497–516, 1957.

[26] K. Komoriya and K. Tanie, “Trajectory design and control of a wheel-type mobile robot using B-spline curve”, *Intelligent Robots and Systems' 89. The Autonomous Mobile Robots and Its Applications. IROS'89. Proceedings., IEEE/RSJ International Workshop on*, IEEE, 1989, pp. 398–405.

[27] A. Takahashi *et al.*, “Local path planning and motion control for AGV in positioning”, *Intelligent Robots and Systems' 89. The Autonomous Mobile Robots and Its Applications. IROS'89. Proceedings., IEEE/RSJ International Workshop on*, IEEE, 1989, pp. 392–397.

[28] W. Nelson, “Continuous-curvature paths for autonomous vehicles”, *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, IEEE, 1989, pp. 1260–1264.

[29] M. Vendittelli, J.-P. Laumond, and C. Nissoux, “Obstacle distance for car-like robots”, *IEEE Transactions on Robotics and Automation* **15**, no. 4 1999, 678–691, 1999.

[30] A. Piazzi *et al.*, “Quintic G/sup 2/-splines for the iterative steering of vision-based autonomous vehicles”, *IEEE Transactions on Intelligent Transportation Systems* **3**, no. 1 2002, 27–36, 2002.

[31] C. G. L. Bianco and O. Gerelli, “Generation of paths with minimum curvature derivative with η^3-splines”, *IEEE Transactions on automation science and engineering* **7**, no. 2 2010, 249–256, 2010.

[32] K. Yang and S. Sukkarieh, “An analytical continuous-curvature path-smoothing algorithm”, *IEEE Transactions on Robotics* **26**, no. 3 2010, 561–568, 2010.

[33] M. Wahde, *Biologically inspired optimization methods: an introduction.* WIT press, 2008.

[34] J. Lehman *et al.*, “Safe mutations for deep and recurrent neural networks through output gradients”, *arXiv preprint arXiv:1712.06563* 2017, 2017.

[35] X. Zhang, J. Clune, and K. O. Stanley, “On the relationship between the OpenAI evolution strategy and stochastic gradient descent”, *arXiv preprint arXiv:1712.06564* 2017, 2017.

[36] E. Conti *et al.*, “Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents”, *arXiv preprint arXiv:1712.06560* 2017, 2017.

[37] J. Lehman *et al.*, “ES is more than just a traditional finite-difference approximator”, *arXiv preprint arXiv:1712.06568* 2017, 2017.

[38] P. J. Angeline, G. M. Saunders, and J. B. Pollack, “An evolutionary algorithm that constructs recurrent neural networks”, *IEEE transactions on Neural Networks* **5**, no. 1 1994, 54–65, 1994.

[39] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies”, *Evolutionary computation* **10**, no. 2 2002, 99–127, 2002.

[40] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, “A hypercube-based encoding for evolving large-scale neural networks”, *Artificial life* **15**, no. 2 2009, 185–212, 2009.

[41] G. I. Sher, “Discover & eXplore Neural Network (DXNN) platform, a modular TWEANN”, *arXiv preprint arXiv:1008.2412* 2010, 2010.

[42] Revere. (2018). OpenDLV - A modern microservice-based software ecosystem for self-driving vehicles, [Online]. Available: `https://github.com/chalmers-revere/opendlv` (visited on 2018).

[43] E. Bahceci *et al.* (2011). NEAT C++, [Online]. Available: `http://nn.cs.utexas.edu/?neat-c` (visited on 2018).

[44] A. Mihály, B. Nemeth, and P. Gáspár, “Integrated vehicle control of in-wheel electric vehicle”, **42** Jan. 2014, 19–25, Jan. 2014.

[45] H. B. Pacejka and E. Bakker, “The magic formula tyre model”, *Vehicle System Dynamics* **21**, no. sup001 1992, 1–18, 1992. DOI: `10.1080/00423119208969994`. eprint: `https://doi.org/10.1080/00423119208969994`. [Online]. Available: `https://doi.org/10.1080/00423119208969994`.

[46] S. Baluja, “Evolution of an artificial neural network based autonomous land vehicle controller”, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **26**, no. 3 1996, 450–463, 1996.

[47] G. Markkula, O. Benderius, and M. Wahde, “Comparing and validating models of driver steering behaviour in collision avoidance and vehicle stabilisation”, *Vehicle system dynamics* **52**, no. 12 2014, 1658–1680, 2014.

[48] T. Salimans *et al.*, “Evolution strategies as a scalable alternative to reinforcement learning”, *arXiv preprint arXiv:1703.03864* 2017, 2017.

Appendix

A NEAT parameters

The NEAT parameters used as default settings for the evolution of the network in the black-box model.

```
trait_param_mut_prob 0.5
trait_mutation_power 1.0
linktrait_mut_sig 1.0
nodetrait_mut_sig 0.5
weigh_mut_power 2.5
recur_prob 0.00
disjoint_coeff 1.0
excess_coeff 1.0
mutdiff_coeff 0.4
compat_thresh 3.0
age_significance 1.0
survival_thresh 0.20
mutate_only_prob 0.25
mutate_random_trait_prob 0.1
mutate_link_trait_prob 0.1
mutate_node_trait_prob 0.1
mutate_link_weights_prob 0.9
mutate_toggle_enable_prob 0.00
mutate_gene_reenable_prob 0.000
mutate_add_node_prob 0.03
mutate_add_link_prob 0.05
interspecies_mate_rate 0.001
mate_multipoint_prob 0.6
mate_multipoint_avg_prob 0.4
mate_singlepoint_prob 0.0
mate_only_prob 0.2
recur_only_prob 0.0
pop_size 150
dropoff_age 15
newlink_tries 20
print_every 10000
babies_stolen 0
num_runs 1
```

B Vehicle model parameters

The physical vehicle parameters used in the vehicle model, based on estimations of the CFS17 race car.

Mass: $m = 188.0$
Moment of inertia: $I_z = 105.0$
Wheel base: $L = 1.53$
Front wheel axis to center of gravity: $l_1 = 0.756$
Friction coefficient: $\mu = 0.9$
Magic Formula parameters:
$C\alpha = 25229.0$
$c = 1.0$
$e = -2.0$

C Maps

Pictures of the 20 unique shapes of the tracks used for training and testing. To enlarge the set most of the tracks had both big and small variants, where the size was altered while shape and track width were preserved.

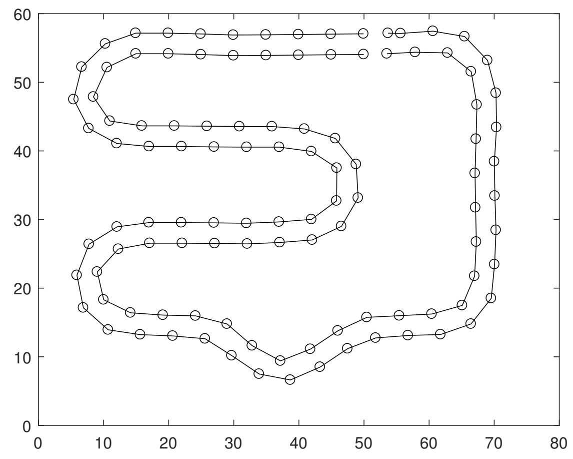


Figure C.1: Track1

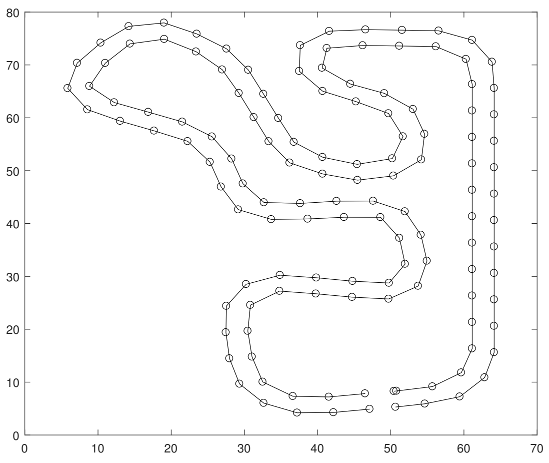


Figure C.2: Track2

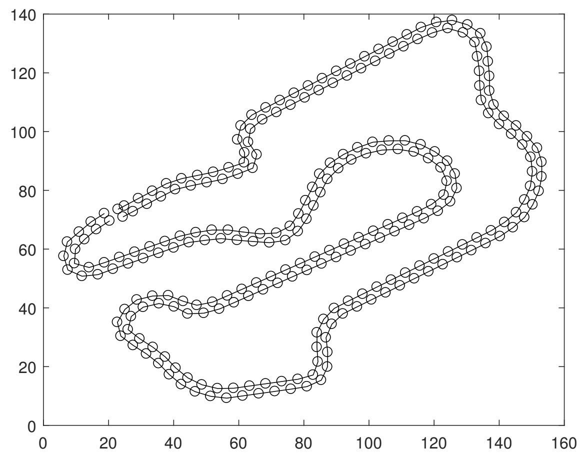


Figure C.3: Track3

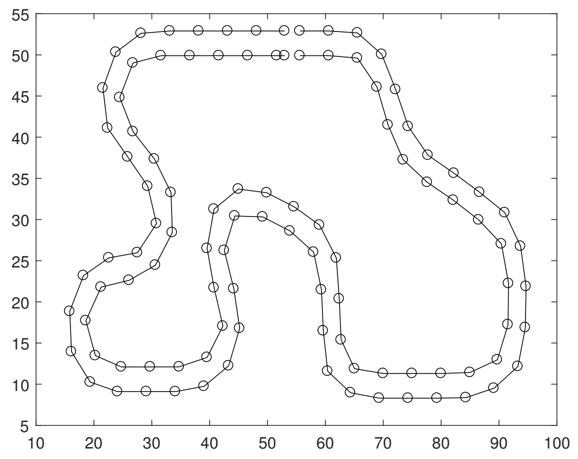


Figure C.4: Track5

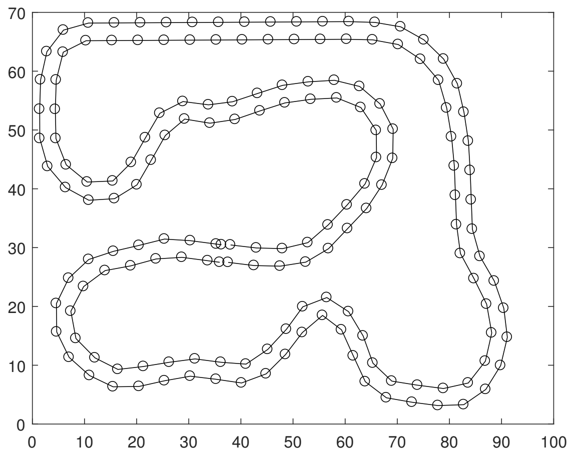


Figure C.5: Track6

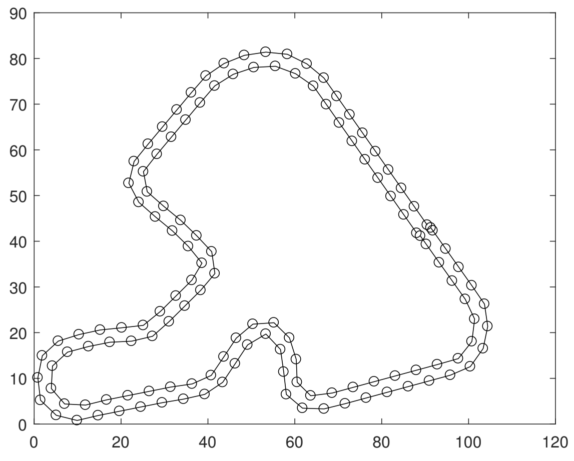


Figure C.6: Track7

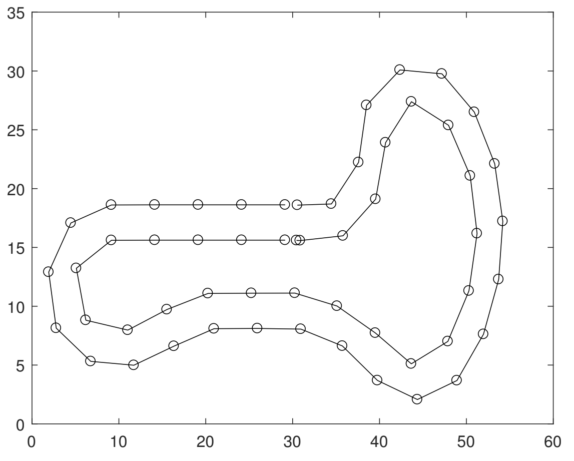


Figure C.7: Track8

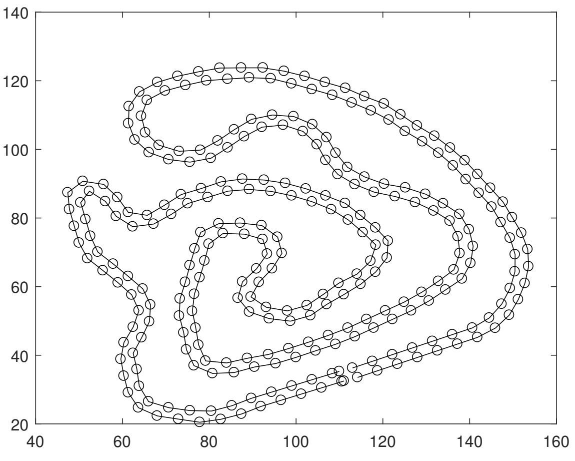


Figure C.8: Track9

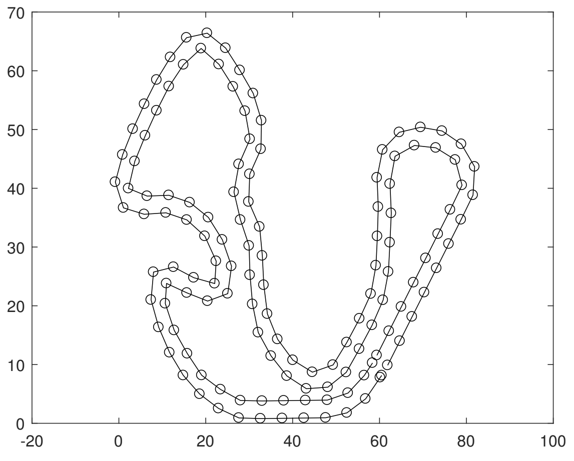


Figure C.9: Track10

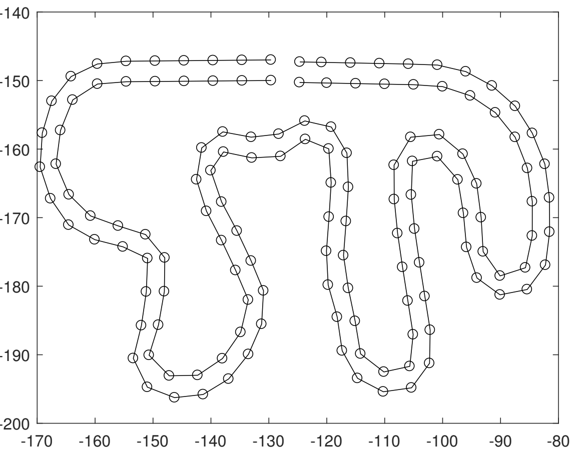


Figure C.10: Track11

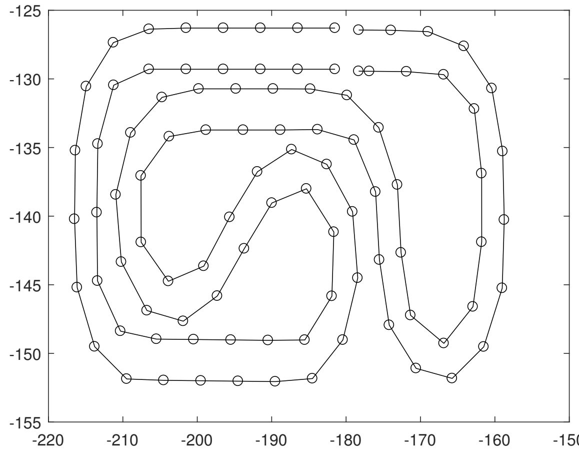


Figure C.11: Track12

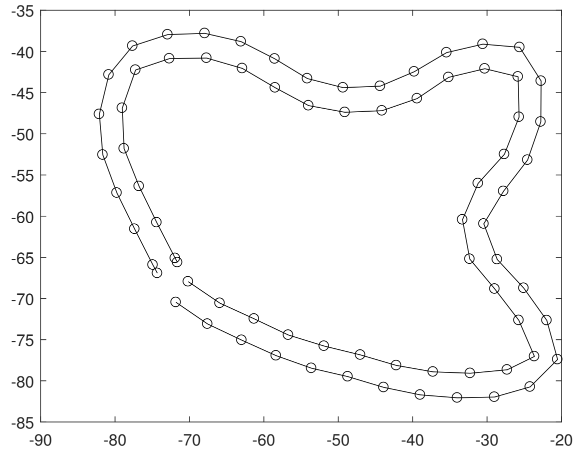


Figure C.12: Track13

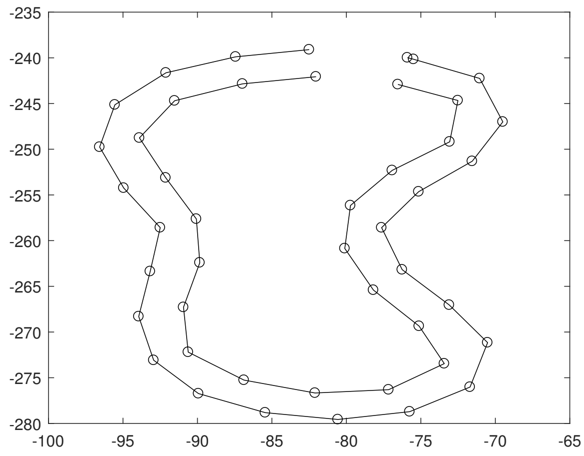


Figure C.13: Track14

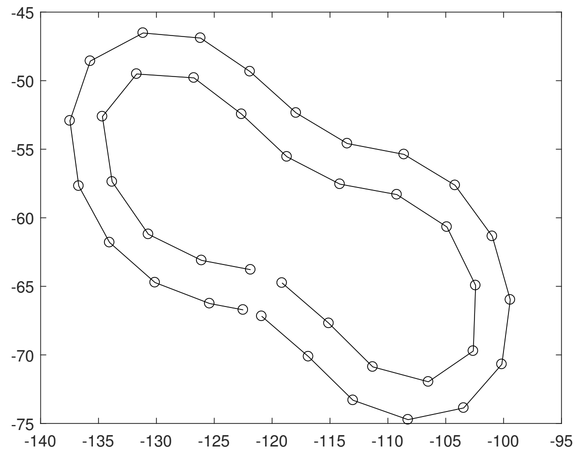


Figure C.14: Track15

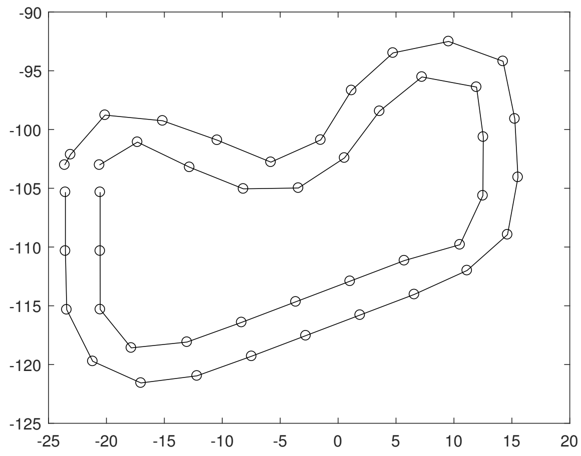


Figure C.15: Track16

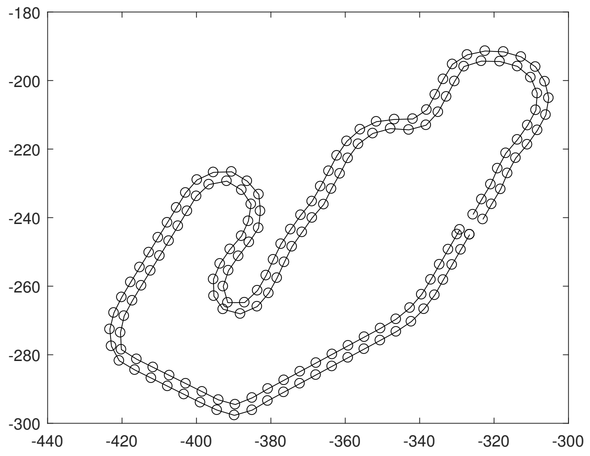


Figure C.16: Track17

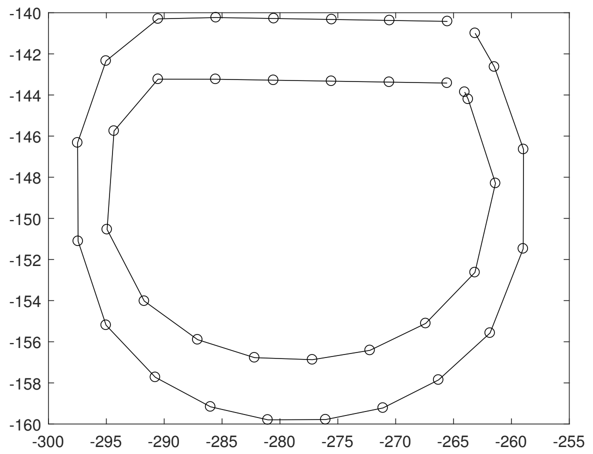


Figure C.17: Track18

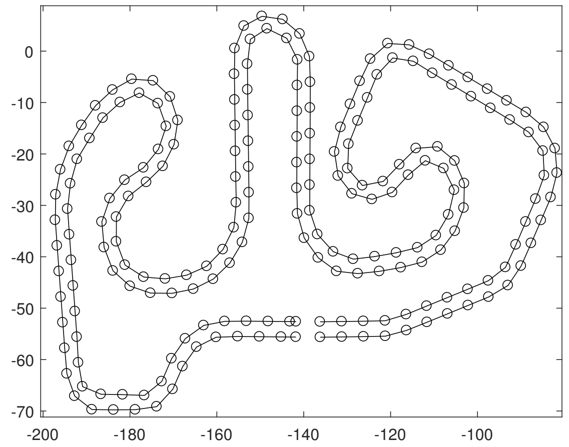


Figure C.18: Track19

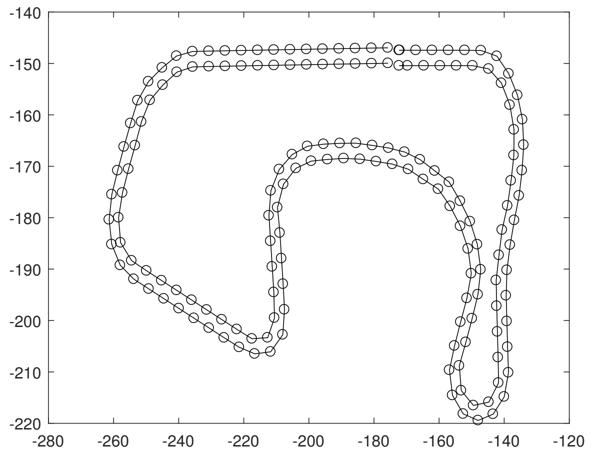


Figure C.19: Track20

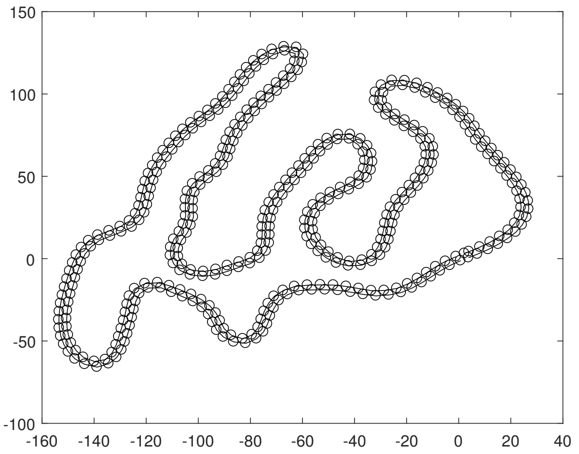


Figure C.20: TrackFSG