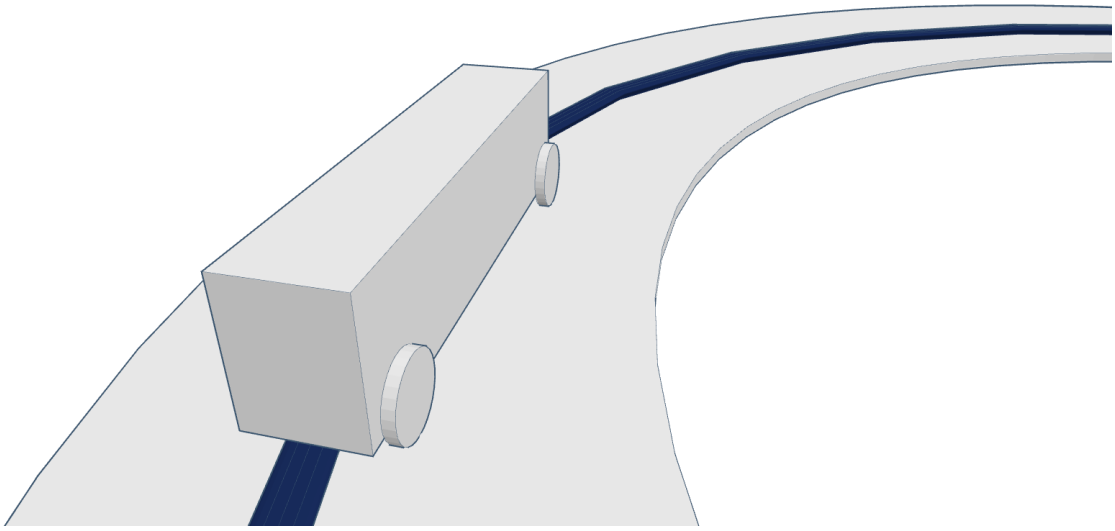
CHALMERS
UNIVERSITY OF TECHNOLOGY



Heavy vehicle path control with neural networks

An evaluation of neural networks for control of heavy vehicles

Master's thesis in Systems, Control and Mechatronics

Viktor Insgård
Lucas Jansson

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

Master's thesis 2018:27

Heavy vehicle path control with neural networks

An evaluation of neural networks for control of heavy vehicles

Viktor Insgård
Lucas Jansson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomus Systems
Adaptive Systems Group
Chalmers University of Technology
Gothenburg, Sweden 2018

Heavy vehicle path control with neural networks
An evaluation of neural networks for control of heavy vehicles
Viktor Insgård
Lucas Jansson

Supervisor(s): Gustaf Johansson and Robert Kull, CPAC Systems AB
Examiner: Peter Forsberg, Adaptive Systems

Master's Thesis 2018:27
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Adaptive Systems Group
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden Telephone +46 31 772 1000

Cover: Illustration of an autonomous truck following a path.

Typeset in LaTeX
Printed by [Name of printing company]
Gothenburg, Sweden 2018

Heavy vehicle path control with neural networks
An evaluation of neural networks for control of heavy vehicles
Viktor Insgård
Lucas Jansson
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Adaptive Systems Group
Chalmers University of Technology

Abstract

This thesis explores the possibility of using neural networks for solving the path control problem, i.e. how to follow a predefined path as closely as possible. Two main approaches are used to achieve this, namely supervised learning and reinforcement learning. The supervised learning approach is based on existing path trackers which are used to generate data for the training procedure. The reinforcement learning uses a genetic algorithm and simulations to evaluate possible solutions. The supervised learning controllers are constructed as feed forward neural networks only, while the reinforcement learning controllers uses a recurrent neural network.

The results shows that neural networks can be trained to solve the path tracking problem, both with supervised and reinforcement learning methods. Both the feed forward networks and the recurrent networks outperform the geometric path trackers. Further, a recurrent network was shown to perform better than a feed forward network, which indicates that the dynamical properties of such networks can be useful in path tracking applications.

Keywords: path control, neural networks, genetic algorithms, autonomous vehicle, heavy vehicle.

Preface

Acknowledgements

Firstly we would like to thank CPAC Systems AB for the opportunity to conduct this thesis at the company as well as the support throughout. We would also like to thank our supervisors Gustaf Johansson and Robert Kull for their invaluable support, feedback and encouragement during the thesis. Finally we would like to thank our examiner Peter Forsberg for his insight and support.

Viktor Insgård and Lucas Jansson, Gothenburg, May 2018

Thesis supervisors: Gustaf Johansson and Robert Kull, CPAC Systems AB
Thesis examiner: Peter Forsberg, Adaptive Systems

Contents

Notation

t	Time
v	Absolute velocity
v_x	Longitudinal velocity
v_r	Reference velocity
ϕ	Steer angle
ψ	Heading angle
F_x	Longitudinal force
F_y	Lateral force
M_z	Torque around the z-axis
d_f	Distance from the front of the vehicle to path
d_c	Distance from the center of the vehicle to path
d_r	Distance from the rear of the vehicle to path
β	Slip angle
P_{F_x}	Gain of F_x controller
P_{M_z}	Gain of M_z controller
θ	Angle to a point on the path ahead
σ	Standard deviation

List of Figures

List of Tables

1

Introduction

The research into autonomous vehicles has gained much interest in recent years, as is evident by the attention the field has gained from some of the major car manufacturers (Audi [1], Toyota [2], Volvo [3], to name a few). The assumption is that the technology will prevent injuries caused by human errors [4], minimize greenhouse gas emissions [5], reduce the total number of needed vehicles [6], and more. For businesses, autonomous vehicles may cut expenditures by removing the need for a driver. It may also open up new possibilities in areas where a human driver may be impractical, due to safety or size concerns.

An important part of an autonomous vehicle is its ability to follow a defined path. To control the vehicle upon this path is known as the path tracking problem. The requirements on the path tracking algorithm of an autonomous vehicle are strict. It is important that the path controller follows the given path closely to stay on track and to avoid potential dangers in the surrounding environment, such as oncoming vehicles or other obstacles.

A number of existing solutions to path tracking problems for autonomous vehicles have been tested by Snider, Jarrod and others [7]. However, these solutions have shown unsatisfactory results in either instability for different velocities or inability to closely follow the predefined trajectory. Additionally, these solutions assumes static system dynamics. When heavy trucks or similar vehicles are considered, system parameters can change quite frequently and be rather large due to e.g. the vehicle being loaded or unloaded.

A field of study that has gained a lot of attention in recent years is that of machine learning and the use of artificial neural networks. Neural networks have seen a wide range of applications, including aiding clinical studies [8] and emission prediction [9], amongst others. The dynamics of front wheel steered vehicles are highly non-linear which suggest that non-linear control methods could be favourable. The versatility of neural networks might therefore be of use in this control application.

By refraining from the use of traditional controllers for the path follower and instead use machine learning and neural networks a more robust and exact controller may be achieved. Since a neural network can better adapt to the behaviour of the truck in question, this would likely result in improved performance when compared to the traditional control methods.

1.1 Purpose

Current path tracking solutions show unsatisfactory results. Investigating new methods aims at finding a new kind of path tracking controller with increased performance with regards to tracking error and robustness. Thus, the purpose of the thesis is to evaluate what benefits using neural networks instead of conventional methods will lead to when solving the path tracking problem.

1.2 Objective

This thesis investigates the possibility to replace existing path tracking methods with neural networks. Comparisons between the performance of the existing methods and the developed algorithms are carried out to quantify the advantages.

1.3 Scope

The structure of the neural networks, learning methods and/or system interfaces and control structure are not predetermined. As such, it is within the scope of this thesis to evaluate how these factors affect the performance of the system and to propose a good, working solution.

1.4 Limitations

High level path planning will not be a part of this project, i.e. it is assumed that a desired path is given as input to the developed control algorithm.

Further, it is assumed that some system states are available for feedback, such as absolute or relative position, velocity and heading. Some disturbances may be added to the state readings to test the robustness of developed algorithms, but state estimation and sensor signals processing is not within the scope of the thesis.

The focus of the thesis is not within vehicle modelling, but in path tracking control. Therefore the models used are kept as simple as possible. This means that more advanced concepts such as tire and suspension dynamics are neglected. The developed solutions may however be tested on more advanced models for verification purposes.

2
Theory

In this chapter the background theory concerning relevant topics is briefly described. The topics described are machine learning, path tracking and control allocation.

2.1 Machine learning

Machine learning refers to a part of computer science studying algorithms for generating programs automatically, rather than writing them by hand. In the broadest sense, machine learning problems may be divided into three different categories: supervised learning, unsupervised learning and reinforcement learning (see Sections 2.1.2, 2.1.3 and 2.1.4).

2.1.1 Artificial neural networks

Artificial neural networks (ANN) are computing systems inspired by the biological neural networks that constitutes the nervous systems of e.g. mammals and insects. They where first proposed by McCulloch and Pitts in 1943 [10].

One of the strengths of ANNs is their ability to approximate any mathematical function. This is known as the *universal approximation theorem*. More specifically it states that a neural network with a single hidden layer can approximate any mathematical function on a subset of $\mathbb{R}^n$, under some conditions [11] (explanations for these concepts are described further down in this chapter). The challenge is how to adjust the parameters of the ANN to achieve the desired behaviour, i.e. find the correct function approximation. The problem formulation of the parameter tuning determines whether it classifies as a supervised, unsupervised or reinforcement machine learning problem.

The biological inspiration ANNs stems from has also influenced the terminology surrounding them. The parameter tuning of ANNs is commonly referred to as *learning* or *training*. From a mathematical standpoints, the term *parameter optimization* might be more fitting.

The artificial neuron

A biological neuron can take any number of inputs and generate a single output [12]. It can be seen as a binary device. Whenever its inputs match a specific requirement it will send an impulse (which can be seen as the value *true*). This impulse may in turn trigger other neurons, which may in turn trigger even more neurons. A single

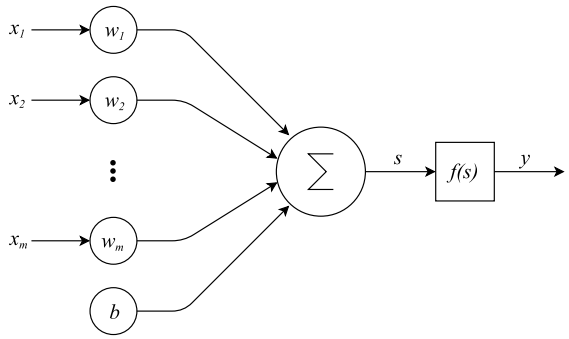


Figure 2.1: *The structure of an artificial neuron.*

neuron can only solve a very simple task, but if many neurons are connected one can achieve very complex behaviours.

Artificial neurons have the same activation mechanism as their biological counterpart. A neuron generates its output from any number of inputs. A neuron may produce binary outputs, but it is more common to configure it to produce continuous outputs. Figure 2.1 illustrates an artificial neuron. Mathematically the output y of a neuron can be calculated by

$$y = f\left(b + \sum_{i=1}^{m} w_i x_i\right) \tag{2.1}$$

where b is the bias, w_i the i:th weight, x_i the i:th input, m the number of inputs to the neuron and f a static function called the *activation function*. The input x_i might be the output of another neuron, or one of the inputs to the ANN itself.

Activation functions

Although it is possible to use any function as an activation function, there are some that are more common. A selection of relevant activation functions are described in this subsection.

Using the biological neuron as inspiration, one might be tempted to use a binary activation function, i.e.

$$f(s) = \begin{cases} 0, & s \leq 0 \\ 1, & s > 0 \end{cases}. \tag{2.2}$$

Using this activation function has a downside, as it will make it difficult to adjust the parameters of the network. Making a small adjustment may cause a new neuron to fire. This may in turn make a larger number of neurons fire later in the network. A small adjustment may therefore cause a significant change in behaviour. It is also possible that a change in the network will have no affect at all on the output. This property of binary neurons makes it difficult to know how to change a network to improve its performance, i.e. train the ANN. These issues can be solved by using continuous activation functions. A common one is the *sigmoid function*, defined as

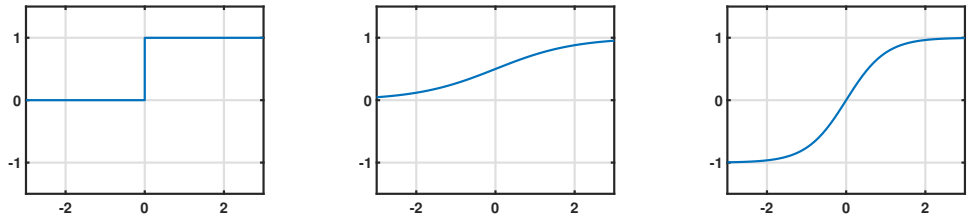
$$f(s) = \frac{1}{1 + e^{-s}}. \tag{2.3}$$

The sigmoid function is similar to the binary activation function for inputs much greater, or less, than 0. For such values the output will be close to 0 or 1. The useful property of the sigmoid function is that it makes small changes in the network detectable in the output, which simplifies the training process. The reason this is the case will be further explained in Section 2.1.2.

Another activation function, that is similar to the sigmoid function, is the *hyperbolic tangens function*, defined as

$$f(s) = \frac{2}{1 + e^{-2s}} - 1. \tag{2.4}$$

A graphical comparison between the described activation functions is shown in Figure 2.2.



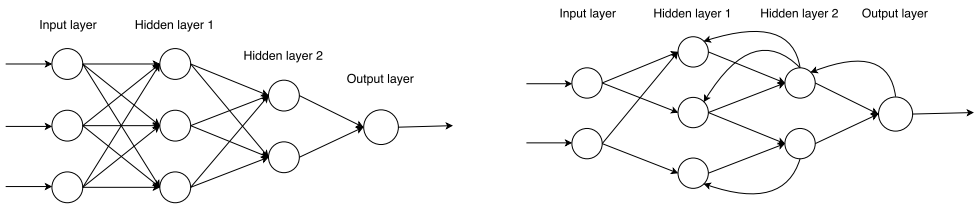
(a) *Binary function.* **(b)** *Sigmoid function.* **(c)** *Hyperbolic tangens function.*

Figure 2.2: *Graphical representations of a few selected activation functions.*

Neural network topologies

The topology of an ANN refers to the structure of the connections between the neurons in a network. Two common topologies are *feed forward neural networks* (FFNN) and *recurrent neural networks* (RNN). An ANN where every neuron has a connection to every other neuron is known as a *fully connected ANN*.

The neurons of an ANN is typically arranged into layers. The input layer is where outside information enters the network. The output-layer is the final layer that generates the outputs from the network. Between the input and output layers there are a number of hidden layers. A *fully connected layer* is a layer where all inputs are fed into every neuron in that layer. Figure 2.3 shows the structure of a few different neural networks.



(a) *A FFNN with fully connected layers.* **(b)** *A non-fully connected RNN.*

Figure 2.3: *Illustration of two different topologies for neural networks.*

In an FFNN all of the neurons are oriented in the same direction. Information flows from the input to the output neurons. This makes an FFNN a static function. It's output will not depend on past inputs.

RNNs allows loops in the network. The loops will give the network a memory function. The input-output relationship will therefore not be static, but rather depend on the current input as well as past inputs. This property can make RNNs useful for applications such as speech recognition, where previous sounds matter as well as the current sound [13]. The dynamical nature of RNNs may also be of use in control and modelling of dynamical systems, as has been demonstrated by Narendra et al. [14].

2.1.2 Supervised learning

Supervised learning uses data in the form of input-output pairs to tune ANNs. Given the available data, the learning algorithm will optimize the parameters of the network so the specific inputs are mapped to the correct outputs. The most commonly used training algorithm for supervised learning is *backpropagation*, which uses *gradient descent* to find the solution.

Gradient descent is an optimization method based on following the gradient of the function to optimize. As an example, consider a simple one dimensional function, such as $y(x) = x^2$, with the gradient $\frac{dy}{dx} = 2x$. Gradient descent requires a specified initial guess of the optimum. In this case, the initial guess might be that the optimal value is $x = 1$. This guess gives $y(1) = 1$ and $\frac{dy}{dx}|_{x=1} = 2$. By looking at the gradient, it is known that $y(x)$ is increasing in the positive x direction, implying that the optimal minimum point lies in the negative x direction. The gradient descent algorithm works as described in Algorithm 2.1.

Make an initial guess
while *Convergence is not reached* **do**
  Calculate the gradient
  Take a step indicated by the gradient
end

Algorithm 2.1: *The gradient descent algorithm.*

The usage of the gradient descent algorithm is one of the reasons using continuous activation functions is preferred over the binary activation function since the gradient is discontinuous. The gradient of the binary activation function is zero for all values, except for $x = 0$ where it is infinite, hence it is impossible to know how to change the parameters of the neuron to improve the performance of the ANN if a gradient based optimization method is used.

The gradient descent then uses backpropagation to relate the weights and biases of a multilayered ANN to its output error. More specifically, it calculates the gradient of the error as a function of the ANNs weights and biases. Once the gradient of the error is known, the gradient descent algorithm can be used to tune the parameters of any FFNN. Backpropagation was first used to tune ANNs in 1986 by Rumelhart et al. [15].

Apart from the standard gradient descent algorithm, there exists a number of similar gradient based algorithms. An example of such an algorithm is *stochastic gradient descent*, which randomly selects only a few weights to optimize each step in the algorithm.

2.1.3 Unsupervised learning

Unsupervised learning does not use input-output pairs, rather it will try to find underlying structures in the provided data. Because if there is no correct observations provided to the learning algorithm, there is no measure of its accuracy, which makes it fundamentally different from supervised and reinforcement learning methods.

2.1.4 Reinforcement learning

Reinforcement learning is used when there is no feasible way to form correct input-output pairs or when it is impractical to generate good training data. As an example, consider a chess robot. It can be difficult to determine whether an individual move in the game is good or bad. Therefore, constructing correct input-output pairs might not be reasonable, which in turn means that it will be hard to use supervised learning algorithms. Instead one might let a group of slightly different chess robots compete against each other. The robots with the best strategies can then be used as a base to form a new group of robots, ready for the next round of games. Repeating this process will reinforce positive traits in the ANNs controlling the chess robots and improve their performance over time. Section 2.1.5 provides a more in depth explanation of how reinforcement learning may be implemented using *genetic algorithms* (GA).

2.1.5 Genetic algorithms

GAs are a group of evolutionary algorithms, which are stochastic optimization methods inspired by natural selection. This is summarized by Wahde in [12]. In nature an individual is described by either one or a set of *chromosomes*. These chromosomes are what results in the physical properties of the individual. Much like natural selection a GA seeks to transfer information from a successful individual in order to retain the properties that makes said individual successful. Practically this is done by storing the chromosome as a string of numbers, which can be of any type, depending on the task at hand. The chromosomes are then given a comparative score, known as *fitness*, calculated from a fitness function. The *fitness function* is formulated in such a way that it favours wanted behaviour whilst penalizing unwanted behaviour. A simple example would be to minimize a mathematical function, where the chromosome represents the variables of the function. Algorithm 2.2 gives a quick overview of how a GA works.

while *End is not reached* **do**
  for *All individuals in population* **do**
    Score and record the fitness of each chromosome in the population
  end
  for *Size of next generation* **do**
    Select individuals according to chosen selection method
    Generate children, cross over with a probability of p_c, otherwise keep the parent chromosomes as before
    Mutate the chromosomes with a probability of p_{mut}
  end
end

Algorithm 2.2: *A general overview of a genetic algorithm.*

Selection methods

There exists several different ways to select the chromosomes that get to carry on their information to the next generation. The simplest way to carry out this selection is to randomly select two chromosomes from the population, where each chromosome has the same probability to be selected. A potential drawback of this method is that due to the randomness of the selection there exists a probability that a chromosome with a higher fitness score is discarded in favour of a low scoring chromosome, resulting in a slower convergence. Other methods utilizes the fitness as a likelihood in order to aid in decreasing the time of convergence for the optimization. A common method to take the chromosomes fitness into consideration is *roulette-wheel selection*. Roulette-wheel selection chooses the two potential individuals by giving each individual a probability to be selected such as $p_{select} = \frac{\text{fitness of individual}}{\text{total fitness of population}}$. However a selection method which always favours the fittest individual has an increased probability to get stuck at a local optimum.

Crossover

If the chromosomes are represented by two arrays of numbers, the crossover function swaps parts of the chromosomes between two selected chromosomes. This can be implemented in a few different ways but the simplest method is the single-point crossover function where a point in the array is selected and the data after that point is swapped between the two chromosomes. Another way to implement this function is to select and swap several points in the data array representing the chromosome. This is known as uniform crossover. However, for smaller populations, crossover may result in a particular chromosome spreading too fast and thus the population may converge to a local optimum and therefore never find the global optimum. To circumvent this unwanted behaviour crossover is typically restricted to only occur with a probability of p_c, allowing other solutions to be explored.

Mutation

Just like in nature, the GA's introduce a possibility of mutating a chromosome. Mutation occurs in order to assure that the population explores the solution space

sufficiently. Mutation is implemented such that for every generation each data entry of a chromosome has a chance to mutate, p_{mut}. For binary encoded chromosomes a mutation would be a change from 0 to 1 or vice versa. If the chromosome data is represented by real numbers a mutation is commonly represented by adding a Gaussian distribution with a mean value of 0 to the gene value.

Evaluation

Just like any other optimization algorithm, the GA needs some definition of what to minimize or maximize. In a GA this is defined by the fitness function. The fitness function determines the difference between a good and bad solution and therefore it should be defined with care.

Since the GA will optimize the given parameters (the genome) according to the fitness function it is important that it reflects the desired behavior. In a simple case, such as finding the minimum of a mathematical function, the fitness function will simply be the function itself. A genome resulting in a higher output value from the mathematical function will not be as fit as one that results a low value. In this way it is easy to evaluate and compare different solution candidates. In a more complex case, such as finding the optimal parameter combination for a PID-controller, it is less obvious how to define the fitness function. In this case it would probably be necessary to run a simulation where the PID-controller controls some system and then evaluate how well the system output signal followed a desired output signal.

2.2 Path control

Path control is a specific application of automatic control regarding controlling agents so that they follow a desired path. The goal is to make the agent follow a specific path, while the distance to the path is minimized. For autonomous vehicles a path controller and a control allocator forms a complete control chain, i.e. a mapping from states $\mathbf{x}$ to actuator commands $\mathbf{u}$. Control allocation is described in Section 2.3.

To give an overview of how path tracking control can be carried out, this section will describe two geometric path tracking methods, namely the *pure pursuit* and the *Stanley method*.

2.2.1 Pure pursuit

Examples of pure pursuit and its usefulness can be seen in the papers written by Snider et al. [7] and Amidi et al. [16], among others. The idea behind this control method is quite simple. A "look ahead" distance l_d is used to find a goal point further down the path. The lateral distance e between this point and the heading vector is then used to calculate a steering angle. Amidi et al. used a fixed look ahead distance l_d, which makes the pure pursuit a pure proportional controller, while Snider et al. proposed that l_d should be assigned as a function of the vehicle's velocity. The

mathematical expression for the steering angle is defined by

$$\phi(t) = \frac{2}{l_d^2} e(t) \tag{2.5}$$

where $\phi(t)$ is the steering angle. Figure 2.4 shows a geometric interpretation of the pure pursuit controller. For a mathematical derivation of this control method, the reader is directed to the paper by Amidi et al. [16].

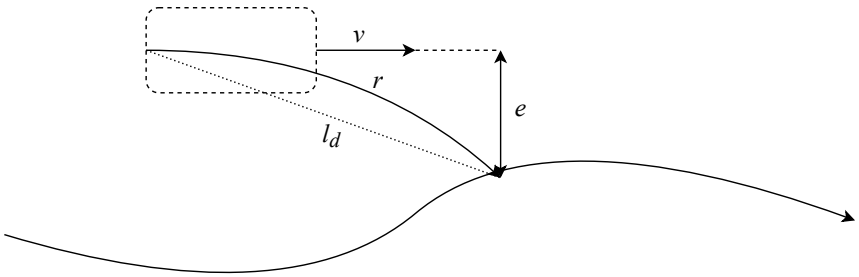


Figure 2.4: *Geometrical relationships used in the pure pursuit path tracking method. In this figure, r is the arc the rear wheels will follow with the current steering angle.*

2.2.2 The Stanley method

The Stanley method is a path tracking method that was first used by the vehicle "Stanley" in the DARPA grand challenge in 2005 [17], a desert race for fully autonomous vehicles. The control method uses a simple heading error ψ_e formulated as

$$\psi_e(t) = \psi(t) - \psi_p(t) \tag{2.6}$$

where ψ is the yaw angle of the vehicle and ψ_p is the angle of the path. Hence, the steering error is simply the difference of the heading of the path and the heading of the vehicle. Additionally a distance error e_d between the front wheels and the path is used to formulate the final control law as

$$\phi(t) = \psi_e(t) + \arctan\left(\frac{k e_d(t)}{v_x(t)}\right) \tag{2.7}$$

where k is an adjustable gain and $v_x(t)$ is the longitudinal velocity. Figure 2.5 shows the geometric relationships used in equation (2.7). It can be shown that using this control method provides exponential convergence of the cross track error [17].

2.3 Control allocation

Control allocation is an issue that arises for an over-actuated system, where there is an actuator redundancy compared to the degrees of freedom of the system. The issue is that, due to the actuator redundancy, there may be several combinations of actuator outputs that acquire the desired total output of the system. The allocator divides the control signal to each of the actuators of the system as seen in figure 2.6.

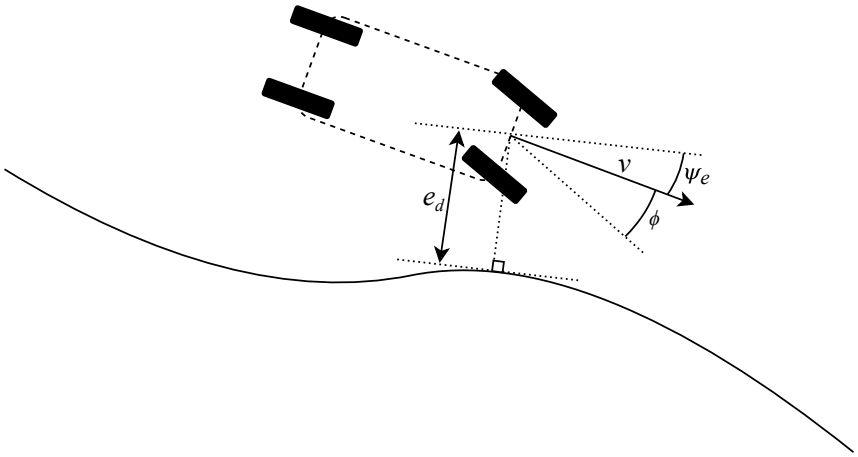


Figure 2.5: *Geometrical relationships used in the Stanley path tracking method.*

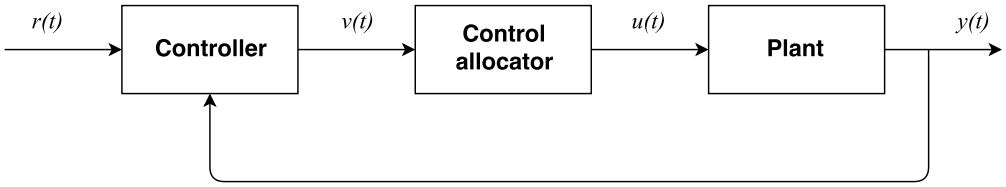


Figure 2.6: *A block diagram structure of a control system with a control allocator.*

The complexity of the allocation varies depending on the method of choosing an actuator combination. A simple way of solving the issue is to select a solution which fulfills the desired control signal $\boldsymbol{v} \in \mathbb{R}^{m\times 1}$ without considering the effectiveness of the solution used compared to other potential solutions. However, this may lead to inefficient use of the actuators, where e.g. actuators may work against each other in achieving the goal. To solve this potential problem, one might instead form an optimization problem to deduce the control signals for the given target and constraints that gives a unique solution. One issue with the more advanced optimizing allocation solvers is the required computational power.

The correlation between the total output and the existing actuators $\boldsymbol{u} \in \mathbb{R}^{n\times 1}$, is given by the effectiveness matrix $\boldsymbol{B} \in \mathbb{R}^{m\times n}$. Therefore the goal is further formulated as

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{v}. \tag{2.8}$$

It is common that the control allocator should also acknowledge the limitations in the actuators. This is done by subjecting equation (2.8) to the actuator constraints

$$\underline{\boldsymbol{u}} \leq \boldsymbol{u} \leq \bar{\boldsymbol{u}}. \tag{2.9}$$

where $\underline{\boldsymbol{u}}$ is a vector of the minimum values that the actuators can supply and $\bar{\boldsymbol{u}}$ is a vector of the maximum values that the actuators can supply.

3

Approach

This chapter gives an overview of the tools, methods and work flow used during this thesis.

The work was split up into two main increments or iterations, with the goal of achieving a controller using different approaches. The goal with the first increment was to create a path controller with supervised learning methods. The developed solution would serve as a base when delving deeper into the project. The purpose of the second iteration was to improve the solution from the first increment by using other learning methods as well as different neural network structures, whose performance would not depend on the provided data, i.e. reinforcement learning methods.

A figure depicting the structure of the system is shown in Figure 3.1. This general structure was used throughout the project and most of the interfaces between the blocks was kept the same. The desired forces used were the longitudinal force F_x, the lateral force F_y, and the torque around the z-axis M_z. The path data was defined as a list of coordinates. The available actuator commands was also kept the same, namely throttle and steering. A more advanced vehicle model with more actuators could have been used, but since the focus was on the path tracking rather than the vehicle modelling, this simpler approach was used. The interface between the path pre-processor and ANN, as well as used system states, was changed as new solutions were developed.

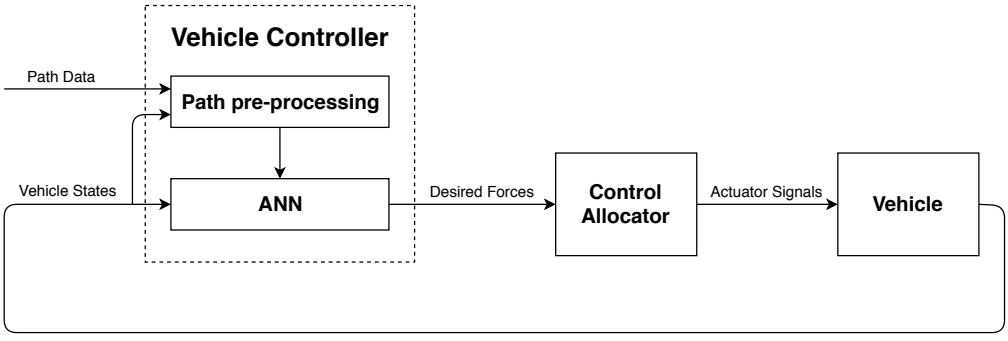


Figure 3.1: *The structure of the control system displayed as a block diagram.*

3.1 Development environment

Much of the work was carried out within the *"robot operating system"*-framework (ROS). ROS is not an actual operating system (OS), rather it is a collection of software and libraries for robotics software development. In ROS's own internal wiki page the term "meta-operating system" is used. The implication of this is that even though ROS runs on Linux systems (Ubuntu and Debian, experimental support for other OSs are available), it provides many features typically provided by an OS, such as low level hardware abstraction, message passing between processes and package management. Additionally, ROS provides visualization tools such as the graph plotting application *qrt* and the 3D-visualization tool *rviz* [18].

ROS uses a server-client model, where the central application (*ROS-core*) acts as a server. Development within ROS is carried out by constructing so called *nodes*, which acts as clients. A node is its own separate program and executes code asynchronously from the ROS-core and other ROS nodes. Nodes may communicate with each other by publishing and subscribing to *topics* and by providing *services*, which can be called on request. Distributing nodes on different computers in the same network is supported, which allows a cluster of computers running ROS to act as one. This is a useful feature in applications where several different electronic control units are used, such as in a modern vehicle.

3.1.1 Simulation environment

Two different simulation environments were used to test and evaluate the results; one simplistic simulator implemented as a C++ class (see Section 3.3) used in the evaluation step of the genetic algorithm as well as a more advanced simulator know as *Gazebo*, which can be integrated with ROS. The reason for using two simulation environments were to have one fast, in terms of execution time, and one which could be used as a verification tool.

Gazebo is a multibody physics simulator. Collisions between objects, friction between surfaces, e.t.c. are all handled by the simulation environment. Models are built by assembling two distinct types of elements. *Links*, which are physical objects such as wheels and chassis, and *joints*, which connects links. Links are defined by their shape, mass and inertia. Joints are defined by their parent link, child links and type of allowed movement, which can be anything from rotation around a vector to movement in a plane. Between each link there must be a joint and between each joint there must be a link. It is not possible to directly connect two links or joints. Hence, the movement of the links is restricted by their corresponding parent joint.

Gazebo's interface to ROS implements services and published topics which can be called or subscribed to. These topics and services allows nodes to read model states. The models can also be controlled by publishing to specific topics that Gazebo is subscribed to. This allows forces and torques to be applied to joints or states of joints and models to be set.

The simplistic C++ simulator is described in Section 3.3.

3.1.2 ANN-frameworks

The deep-learning library *Caffe* [19] was used to train and implement the ANNs developed with supervised learning methods. Contrary to many popular deep learning frameworks, which only supports Python, Caffe also supports development in C++ and MATLAB (for MATLAB versions up to 2015a) as well. The optimization algorithm used for the supervised learning was *stochastic gradient descent* (see Section 2.1.2).

Caffe, however, did not provide sufficient support for the networks with recurrent structures, the RNNs. Thus another support library had to be used to implement networks with this structure. The library that was used instead was *nnetcpp* [20], which as the name suggests is a C++ library. This library was chosen due to its minimalist implementation and the simplicity of using it to design an ANN. Another desired property with this library is that the weights and biasases of the network could easily be accessed and changed, which is required to use the network with the GA, as described in Section 5.1.

3.1.3 GA-framework

In order to improve on the solution obtained with supervised learning, optimization of the ANNs were also done with a GA. In contrast with optimization methods like the ones used for supervised learning, a GA does not need to be told what to optimize for, it only needs a way to evaluate solution candidates (see Section 2.1.5). For this reason, it was used to implement reinforcement learning.

The learning procedure was realized with the assistance of the C++ library *GAlib*, which was developed by Mathew Wall at Massachusetts Institute of Technology [21]. This library utilizes a user specified fitness function in order to evaluate the performances of each network. Since some built in functions were considered unsatisfactory in the sense that they limited the search space, new versions were developed and added. These new functions included a method that handles mutations, as well as the scoring function. The used fitness function is described in Section 3.3.

3.2 Path pre-processing

There are a variety of methods that can be used to represent a path. Some examples included as checkpoints, as line segments, or as different types of splines. This can be an issue when designing a path controller, since these different representations requires different interpreters to generate error signals for the controller to act upon. A solution to this is to split the path interpretation and the control signal generation. This allows seamless changes between different interpreters and controllers, as long as they share a common interface. Throughout this report, the path interpretation block is referred to as the *path pre-processor* (PPP).

The role of the PPP is to interpret the path data, which can vary in size, along with the state vector of the vehicle into a control error on a standardized form. This control error is then sent as input to the ANN controller which uses a fixed size input vector.

The implemented PPP uses a set of two dimensional points in the global reference frame as a path. When calculating an output error, it will use linear interpolation to interpret the path as a set of line segments.

The PPP acts in the global reference frame, i.e. it uses the global pose of the vehicle along with the path defined by global coordinates to generate an output. The output of the PPP is, however, given in the vehicles local coordinate frame for the ANN controller to act upon. If the PPP output was not in the local frame, the size of the ANN would have to be increased to incorporate the transform between reference frames, and the number of inputs would also have to be greater. This would decrease the converge rate of the training procedure, while providing limited benefits.

3.3 Evaluation of results

To be able to measure and compare the performance of different methods, an evaluation equation was used. This evaluation was based on three different groups of metrics: input signals, velocity and distance to the path. To further penalize methods that let the vehicle operate far away from the intended path, desired input signals or desired velocity, each metric was squared. Further, it was also deemed a good quality if as much as possible of the vehicle was on the path at all times, therefore three different distance metrics where used. The measured distances where from the center of the vehicle to the path, from the center of the front axle to the path and lastly, from the center of the rear axle to the path. Mathematically, it was defined as

$$\begin{aligned} \Gamma_c &= \int_{t_0}^{t_f} \mathbf{y}^T(t)\mathbf{W}\mathbf{y}(t)\,dt, \\ \mathbf{y}(t) &= \begin{bmatrix} v_e & F_x & M_z & d_f & d_c & d_r \end{bmatrix}^T \end{aligned} \tag{3.1}$$

for continuous time. For the implementation a discrete version of the evaluation function was defined as

$$\begin{aligned} \Gamma_d &= \Delta t \sum_{n=0}^{N} \mathbf{y}^T[n]\mathbf{W}\mathbf{y}[n], \\ \mathbf{y}[n] &= \begin{bmatrix} v_e & F_x & M_z & d_f & d_c & d_r \end{bmatrix}^T, \end{aligned} \tag{3.2}$$

where v_e is the difference between the reference and actual velocity, F_x is the desired forward force, M_z is the desired torque around the z-axis and d_f, d_c and d_r are the front, center and rear distances respectively as seen in Figure 3.2. $\mathbf{W}$ is a weighting matrix. F_x and M_z are the global forces used as input to the control allocator, as described in Figure 3.1. The parameter values of the weightings used are defined in Table 3.1.

Three test tracks where defined with three different aspects to test in mind. *Track A* demonstrates a lane change maneuver, which could be viewed as the step response equivalence for path tracking controllers. *Track B* is meant to give an idea of the steady state behaviour of a controller. Some path tracking methods have a tendency of cutting corners in curves and this type of track with long curves illustrates this

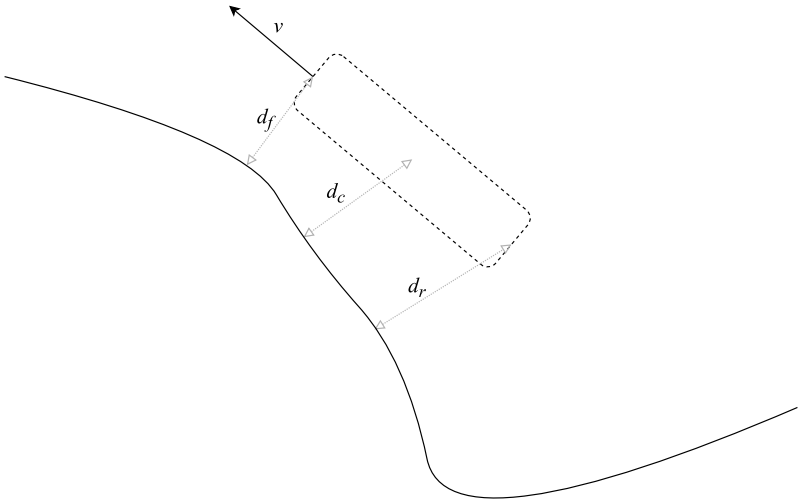


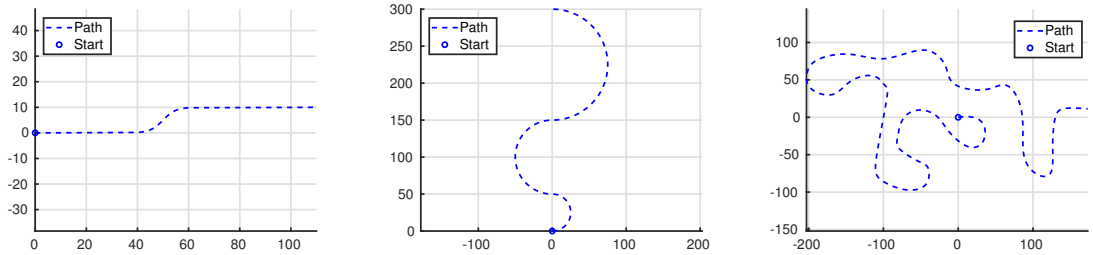
Figure 3.2: *Illustration of the three distances used in the evaluation, namely the front, center and rear distances from the vehicle to the path, which are represented by d_f, d_c and d_r respectively.*

Table 3.1: *Parameters used in the evaluation function.*

Parameter	Value
Δt	1/30
N	path dependent
w_v	1.0
w_{F_x}	10^{-11}
w_{M_z}	10^{-1}
w_{d_f}	1.0
w_{d_c}	1.5
w_{d_r}	1.0

behaviour well. *Track C* is meant to represent a more realistic path, with more varied path characteristics. The test tracks are shown in Figure 3.3.

The controller evaluation tests where carried out in two simulation environments, namely in the Gazebo simulation environment as well as a custom C++ simulator. The purpose of using two different simulations was to have one environment which could act as a verification tool, and one which would be fast to execute. The implementation of the C++ simulation was based on a simple bicycle model. Bicycle models can be found in many sources, but this one in particular was taken from a paper by Kong et al. [22] and then slightly modified. The system of differential



(a) *Track A.* **(b)** *Track B.* **(c)** *Track C.*

Figure 3.3: *Plots of the three tracks used in the evaluation of different controllers.*

equations is described by

$$\dot{x} = v\cos(\psi + \beta) \tag{3.3a}$$
$$\dot{y} = v\sin(\psi + \beta) \tag{3.3b}$$
$$\dot{\psi} = \frac{v}{l_r}\sin(\beta) \tag{3.3c}$$
$$\dot{v} = \frac{\tau}{m} - \mu v \tag{3.3d}$$
$$\beta = \arctan\left(\frac{l_r}{l_f + l_r} + \tan(\phi)\right) \tag{3.3e}$$

were x and y are the 2D coordinates of the vehicle, ψ is its heading, v its velocity and β its slip angle. l_r is the length from the center of gravity to the rear axle of the vehicle and l_f is the length from the center of gravity to the front axle. τ is the longitudinal force, m is the mass, μ is a friction coefficient and ϕ is the angle of the front wheel [22].

The set of differential equations where discretized using a first order Euler approximation, resulting in

$$x[n+1] = x[n] + \Delta t \cdot v[n]\cos(\psi[n] + \beta[n]) \tag{3.4a}$$
$$y[n+1] = y[n] + \Delta t \cdot v[n]\sin(\psi[n] + \beta[n]) \tag{3.4b}$$
$$\psi[n+1] = \psi[n] + \Delta t \cdot \frac{v[n]}{l_r}\sin(\beta[n]) \tag{3.4c}$$
$$v[n+1] = v[n] + \Delta t \cdot \left(\frac{\tau[n]}{m} - \mu v[n]\right) \tag{3.4d}$$
$$\beta[n] = \arctan\left(\frac{l_r}{l_f + l_r} + \tan(\phi[n])\right). \tag{3.4e}$$

In addition to the bicycle model itself, an inertia was added to the steering input, to improve the realism to the model. The time discrete expression for the steering angle was therefore defined as

$$\phi[n] = 0.25\phi[n-1] + 0.75\phi_d[n] \tag{3.5}$$

where $\phi_d[n]$ is the desired input steering angle.

4

Supervised learning path controller

Initially, an ANN controller was developed with supervised learning methods. The purpose of this was to show that a neural network could be generated to solve the path tracking problem. Additionally, the trained ANN would provide insights into suitable network structures when using an alternative training method, as is done in Chapter 5.

4.1 Data acquisition

Initially a controller would be developed using supervised learning. Supervised learning requires substantial amounts of data that represents the scenarios that the ANN should be able to manage. A way to gather this data would be to record the actions of a human driver for a large set of routes. However it was deemed to be unfeasible to achieve a sufficient data set within the thesis time line. The data was instead recorded from simulations in the Gazebo environment using a setup with two different geometric path controllers. These provided the desired longitudinal and lateral forces in the local frame, F_x and F_y, as well as the torque around the local z-axis, M_z. A control allocator responsible for calculating the actuator commands was also added to the control chain.

4.1.1 Pure pursuit controller

One of the controllers used to generate the training data was a variant of the pure pursuit controller, with a fixed look ahead distance and the angle to the path intersection rather that the lateral distance.

The geometric path trackers described in Section 2.2 only considers the steering of the vehicle, i.e. how to control the angle of the front wheels. Decoupling the steering from the velocity control may be desired to decrease the complexity of the complete control scheme of the vehicle. It does, however, neglect the interaction between the two. In a sharp curve a lower speed than desired may be required to stay on course. To add the consideration of velocity to the controller, a proportional regulator was included to control it. The velocity and steering are still decoupled in this case, but once neural networks are considered, this is not necessarily the case.

With the addition of the velocity control, the mathematical expression for the

control law was defined according to

$$\begin{aligned} F_x &= P_{F_x}\left(v_{ref} - v_x\right) \\ M_z &= P_{M_z}\theta \end{aligned} \tag{4.1}$$

where P_{F_x} is the velocity controller gain, v_{ref} is the reference velocity, v_x is the forward velocity of the vehicle in its local reference frame, P_{M_z} is the steering controller gain and θ is the angle to the path 10 m ahead, seen from the center of the vehicle. The gains where determined experimentally and set to $P_{F_x} = 5000$ and $P_{M_z} = 1$.

4.1.2 Stanley method controller

In addition to the pure pursuit controller, data was also recorded with a controller based on the Stanley method. This controller was also paired with a proportional controller for the velocity, since it also only controls the steering angle of the front wheel (the motivation for this is presented in Section 4.1.1). The mathematical expression for the complete controller could therefore be defined as

$$\begin{aligned} F_x &= P_{F_x}\left(v_{ref} - v_x\right) \\ M_z &= \psi_e + \arctan\left(\frac{ke_d}{v_x}\right) \end{aligned} \tag{4.2}$$

where the gain parameter k was set to 1.0.

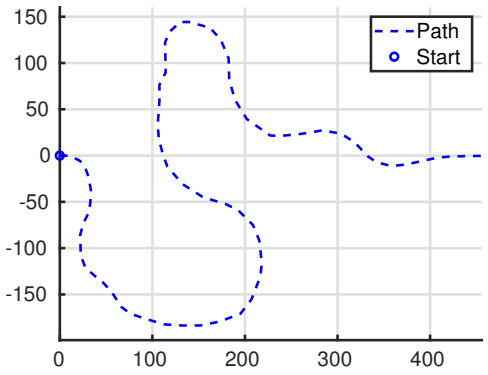
4.1.3 Collecting the data

Simulations in Gazebo using the two controllers were run for four different tracks, shown in Figure 4.1. A simulation was run twice for each track, one in each direction. To enable for the ANN to learn all the different modes of the controller, the reference speed was changed randomly every fourth second and set back to zero every thirtieth second. The changes in reference speed was drawn from a uniform distribution in the range 0 to 10 m/s. The simulations where run until the vehicle had reached the end of each track. In total, roughly 90 000 training samples were recorded from each controller, which was deemed a sufficient data set.

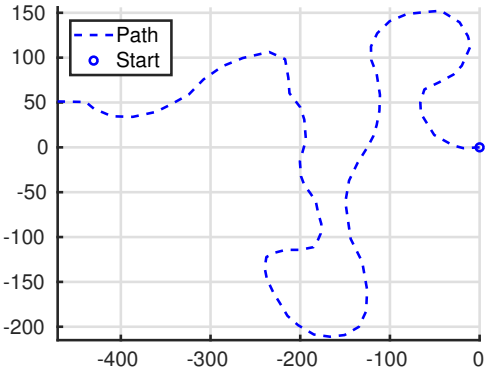
4.2 ANN controller

The type of network that was used to imitate the controller was a FFNN, as shown in Figure 2.3a. As such, the ANN only uses current information to calculate an output, which can be a drawback in control applications. However, for this particular scenario the goal was to imitate the behaviors of a static controller, which also lacks any dynamic behaviour. Due to the similarities in the attributes of the FFNN and the controller it was considered to be possible for the ANN to perform similarly to the controller.

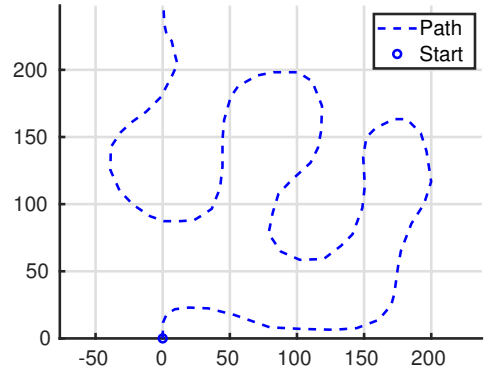
Two tests where carried out to test this hypothesis, one for each controller type. For the pure pursuit data an ANN with one hidden layer containing six neurons was constructed. This small network was found to be able to imitate the controller



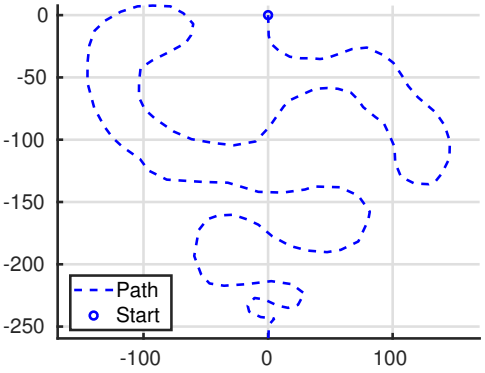
(a) *First path used to generate training data.*



(b) *Second path used to generate training data.*



(c) *Third path used to generate training data.*



(d) *Fourth path used to generate training data.*

Figure 4.1: *The paths used to generate training data for the supervised learning methods.*

sufficiently close. The small size of the network being sufficient is likely due to the simple nature of the pure pursuit control law, Equation (2.5). For the data generated with the controller based on the Stanley method, an ANN with two hidden layers containing 9 neurons each was generated. The requirement of a larger network is likely due the more complex control law used to generate the data. The sizes of the ANNs where chosen so that the validation step in the training process yielded a root mean square error of about 0.01. The network structure used for these ANNs are summarized in Table 4.1.

For the training process, the ordering of the acquired data was randomized and split into two different data sets. Two thirds of the data set were used for the parameter optimization of the ANN. The remaining third was used for the validation step of the training procedure. The validation step was used to confirm the performance of the ANN on a different data set, i.e. that the mapping from inputs to outputs was done correctly. Note that a network that passes this validation step not necessarily performs well when controlling a vehicle, it only concerns the mapping of values discussed earlier.

Table 4.1: *Summary of the two constructed ANNs trained with supervised learning.*

	Pure pursuit ANN	**Stanley method ANN**
Inputs	v_x, v_r, θ	v_x, v_r, ψ_e, e_d
First layer	6 neurons	9 neurons
Activation	tanh	tanh
Second layer	3 neurons (output)	9 neurons
Activation		tanh
Third layer		3 neurons (output)

4.3 Results

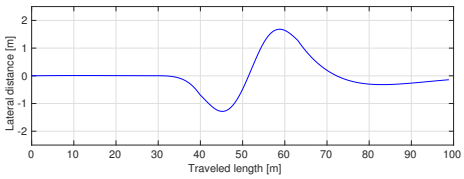
The Stanley method controller, the pure pursuit controller and the generated ANN controllers where tested on the three test tracks, both in the Gazebo simulation environment and in the simpler custom C++ simulation.

Figure 4.2 shows the lateral distance between the center of the vehicle and the path during the C++ simulations run with the pure pursuit controller and its ANN counterpart. Both of the controllers manage to follow each of the paths. Figure 4.2a and 4.2b illustrates the tendency of cutting corners in sharp curves for this controller type.

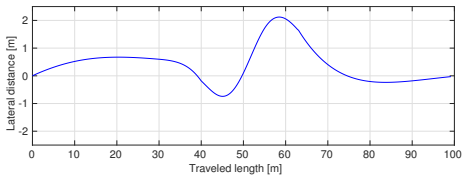
The pure pursuit controller performs better in the wide long curve scenario of track B, where it is only drifting a way from the path initially. The cause of this is probably the nature of the preview distance. When it is at the end of one of the curves and about to change turn direction, it will look at the path in the next curve and follow that path, even though it is still in the previous curve.

Figure 4.3 shows the results from the simulation run with the Stanley controller and its corresponding ANN controller. Both of them also manages to follow all of the paths. On track A (Figure 4.2a and 4.3a) the difference between the two simple control methods is hard to spot. A reason for this is a slight similarity between the two. The Stanley method is based on the position of the front axle of the vehicle. Looking from the center of the vehicle, this is in some sense a preview distance, just like the one used in the pure pursuit. This is also evident in Figure 4.3c. In theory, the Stanley method should not have any remaining error, since the last arctan-term in the control law should steer the front wheels towards the path. However, this will only put the front wheels on the path, the rest of the vehicle will lag behind and follow the more narrow turn radius of the rear wheels. Therefore a remaining control error is to be expected when the center of the vehicle is considered.

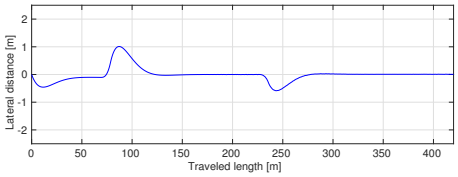
By comparing the ANN controllers trained on the data from the pure pursuit and Stanley method controllers respectively, it is evident that the ANN controllers behaves similarly to the controllers they are based upon. However, it is clear that the performance of the ANN controllers is worse than the traditional control methods. The likely cause of this is a combination of several factors. The size of the ANNs may not have been large enough, the learning algorithm may not have been properly tuned or the training data may not have been good enough. Many of these issues could probably be solved, but the focus of this thesis was not to create ANNs that mimic existing control schemes, rather it was to find new path tracking methods.



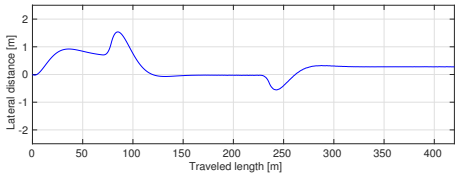
(a) *Lateral distance from the path to the center of the vehicle when following test track A using pure pursuit.*



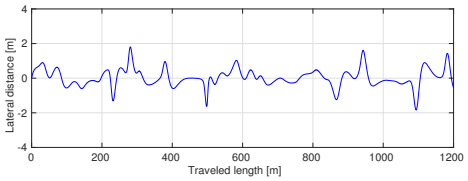
(b) *Lateral distance from the path to the center of the vehicle when following test track A using the ANN generated from pure pursuit.*



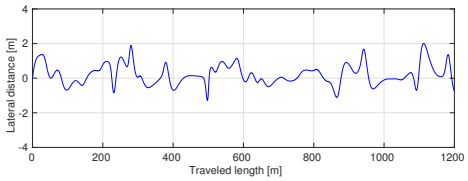
(c) *Lateral distance from the path to the center of the vehicle when following test track B using pure pursuit.*



(d) *Lateral distance from the path to the center of the vehicle when following test track B using the ANN generated from pure pursuit.*



(e) *Lateral distance from the path to the center of the vehicle when following test track C using pure pursuit.*

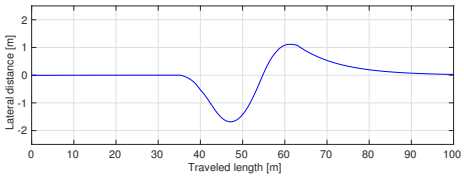


(f) *Lateral distance from the path to the center of the vehicle when following test track C using the ANN generated from pure pursuit.*

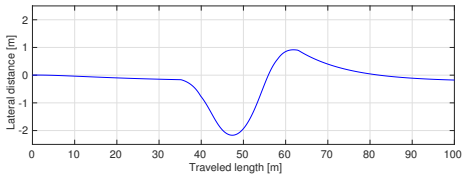
Figure 4.2: *Lateral distance from the path to the vehicle when using the pure pursuit controller and the ANN controller trained with the pure pursuit generated data, for each of the test tracks. These simulations where run with the C++ simulation.*

Therefore more effort was put into exploring alternatives to these controllers, rather than optimizing the existing ones.

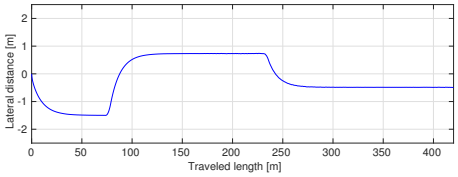
From this stage there were mainly two apparent approaches in order to improve the controller, namely to either use different controller data (as mentioned earlier) to train the ANN or to change learning method. One limitation with supervised learning is that the recorded data decides the optimum of the ANN, i.e. the ANN will at best exactly mirror the recorded data used during the training. Therefore it was deemed to be more beneficial to use a different learning method rather than record new data. This would allow training of a controller that was better than either of the pure pursuit or Stanley method controllers. The learning method and how it was implemented as well as its results are further described in Chapter 5.



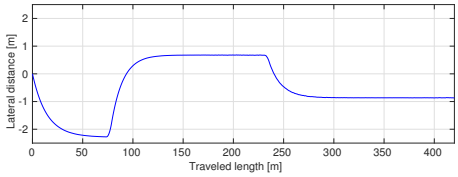
(a) *Lateral distance from the path to the center of the vehicle when following test track A using the Stanley method.*



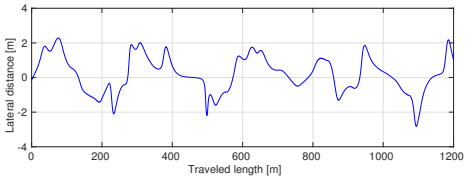
(b) *Lateral distance from the path to the center of the vehicle when following test track A using the ANN generated from the Stanley method.*



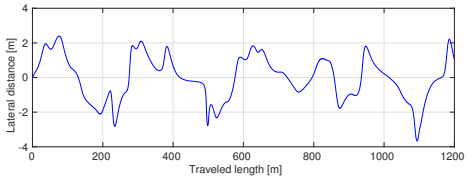
(c) *Lateral distance from the path to the center of the vehicle when following test track B using the Stanley method.*



(d) *Lateral distance from the path to the center of the vehicle when following test track B using the ANN generated from the Stanley method.*

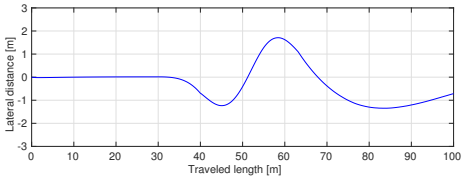


(e) *Lateral distance from the path to the center of the vehicle when following test track C using the Stanley method.*

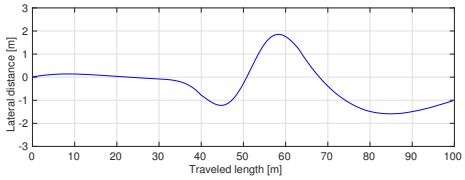


(f) *Lateral distance from the path to the center of the vehicle when following test track C using the ANN generated from the Stanley method.*

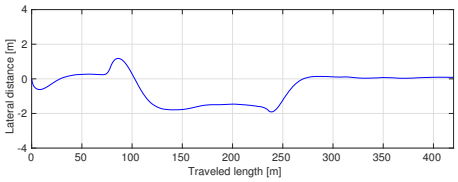
Figure 4.3: *Lateral distance from the path to the vehicle when using the Stanley method controller and the ANN controller trained with the Stanley generated data, for each of the test tracks. These simulations where run with the C++ simulation.*



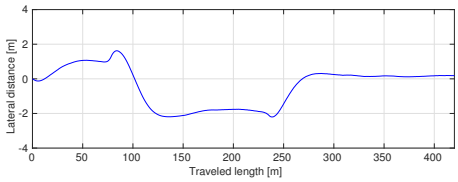
(a) *Lateral distance from the path to the center of the vehicle when following test track A using pure pursuit.*



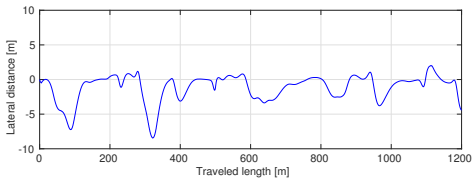
(b) *Lateral distance from the path to the center of the vehicle when following test track A using the ANN generated from pure pursuit.*



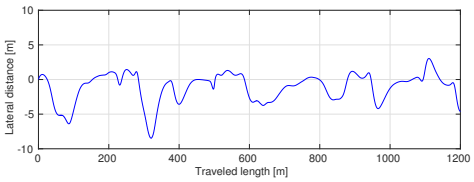
(c) *Lateral distance from the path to the center of the vehicle when following test track B using pure pursuit.*



(d) *Lateral distance from the path to the center of the vehicle when following test track B using the ANN generated from pure pursuit.*

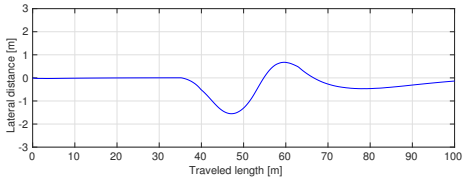


(e) *Lateral distance from the path to the center of the vehicle when following test track C using pure pursuit.*

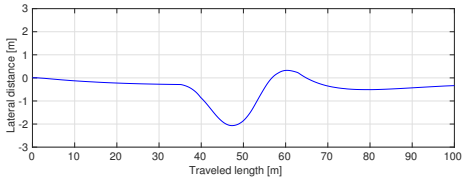


(f) *Lateral distance from the path to the center of the vehicle when following test track C using the ANN generated from pure pursuit.*

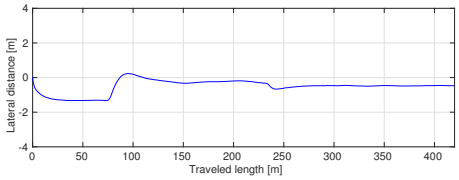
Figure 4.4: *Lateral distance from the path to the vehicle when using the pure pursuit controller and the ANN controller trained with the pure pursuit generated data, for each of the test tracks. These simulations where run with the Gazebo environment.*



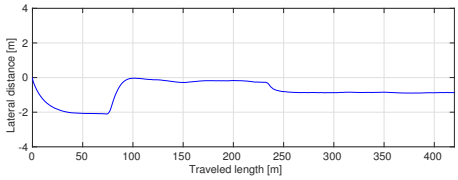
(a) *Lateral distance from the path to the center of the vehicle when following test track A using the Stanley method.*



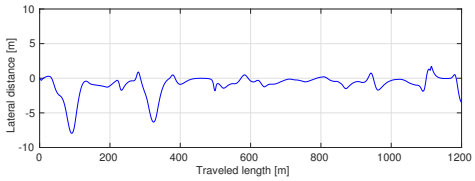
(b) *Lateral distance from the path to the center of the vehicle when following test track A using the ANN generated from the Stanley method.*



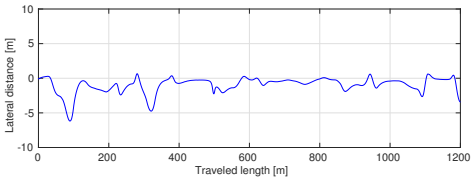
(c) *Lateral distance from the path to the center of the vehicle when following test track B using the Stanley method.*



(d) *Lateral distance from the path to the center of the vehicle when following test track B using the ANN generated from the Stanley method.*



(e) *Lateral distance from the path to the center of the vehicle when following test track C using the Stanley method.*



(f) *Lateral distance from the path to the center of the vehicle when following test track C using the ANN generated from the Stanley method.*

Figure 4.5: *Lateral distance from the path to the vehicle when using the Stanley method controller and the ANN controller trained with the Stanley generated data, for each of the test tracks. These simulations where run with the Gazebo environment.*

5

Reinforcement learning path controller

The controller obtained with supervised learning can only be as good as the controller that generated the training data. To improve it further, a method involving a GA was formulated. This chapter describes such a method and the results obtained from it.

5.1 Implementation of the GA

The task of the GA was to optimize the controller to keep the vehicle as close to the path as possible, i.e. to minimize deviation from the path. In addition, the controller should control the speed of the vehicle, all in such a way that neither the steering nor the throttle use excessive commands.This is described in more detail in Section 5.1.2.

5.1.1 Genome formulation

In order to use the GA for the ANN controller the ANN must first be expressed as a genome. A genome was formulated as an array of real values, where each value corresponds to a weight or bias of the ANN. Thus an initial population, i.e. several genomes, could easily be generated and evaluated as described in Section 5.1.2. The individuals of the population were initialized by assigning each gene a random number drawn from a uniform distribution. The parameters used in the GA are shown in Table 5.2.

5.1.2 Genome evaluation

To speed up the training procedure, the simpler C++ simulator was used, as opposed to the Gazebo model (see Section 3.3).

The states that were recorded for the evaluation were the output torque around the z-axis (M_z), the forward force (F_x), the error velocity of the vehicle ($v_e = v_r - v$) and three distances, namely the front (d_f), center (d_c) and rear (d_r) distances from the vehicle to the path. The measured distances are illustrated in Figure 3.2. The intention with these measurements were to promote a controller that could keep the vehicle on track in the correct angle, i.e. facing down the path, at a desired velocity,

while consuming minimal energy. The fitness function itself is defined in Equation (3.2).

The weights used in the fitness function where developed and changed frequently throughout the thesis work. The final weights used are defined in Table 5.1. Some observations were made on what constitutes a good set of weights. Most importantly, if the velocity weight was chosen to be too small, the controller would get stuck at a local optimum where the vehicle would stand completely still. Also, the weights for F_x and M_z were set low. The purpose of them were not to encourage driving where the vehicle went off track because using the steering was too expensive. Rather it was to discourage excessive use of large control signals, such as sudden changes between maximum steering wheel angles.

Table 5.1: *The weights used in the fitness function of the GA.*

Weight	Value
w_{M_z}	10^{-1}
w_{F_x}	10^{-11}
w_{v_e}	1
w_{d_f}	1
w_{d_c}	1.5
w_{d_r}	1

The simulations were run on a path that was designed to test the controller in every possible system state, i.e. every possible driving situation. More specifically, the path was built with both left and right curves of varying radius, as well as long straight parts. Figure 5.1 shows a plot of the used path. The total length of this track is almost 1 km. In theory several shorter paths could have been used just as well. However, from the point of view of the C++ implementation of the algorithm, using a longer path required less effort. The reference velocity was changed between zero and a positive non-zero value every fourth second. This was required for the ANN to learn how to control the throttle to match the reference speed.

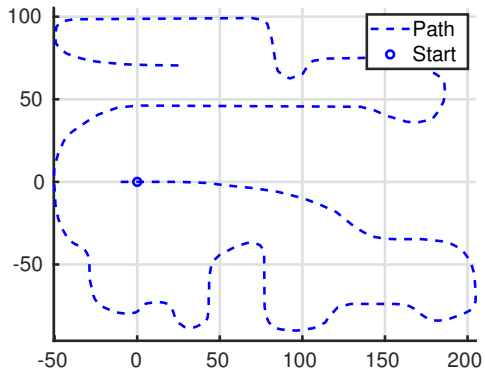


Figure 5.1: *The path used to evaluate genomes in the GA.*

5.1.3 GA parameters

Other than an evaluation function, the GA requires a selection, a crossover, and a mutation method. As mentioned in Section 2.1.5, there are several alternatives for each of these methods. The selection method used was a roulette wheel selection model. If the selected individuals were to crossover, the uniform crossover method was used. For each individual value of a genome a mutation was made with a given probability. If a genome was to be mutated, a random value drawn from a zero mean normal distribution was added to the genomes current value. The specific parameters used can be found in Table 5.2.

Table 5.2: *The parameters used in the GA. The mutation standard deviation was changed from large to small as the training progressed.*

Parameter	**Value**
Individuals	100
Init. range	± 0.01
Crossover likelihood	90 %
Mutation likelihood	1 %
Mutation σ	1 - 0.01

5.2 Network structure

Based the networks developed during the supervised learning part of the project, a network structure was designed with the purpose of improving the previous controller. The hypothesis was that a controller that could use the strength of both a pure pursuit controller and a Stanley method controller would be better than any one of them alone. With this in mind an FFNN with two hidden layers was generated. The structure of the ANN is shown in Table 5.3. The inputs used in both of the simple controllers was added. The ANN was then trained with the GA.

Table 5.3: *Summary of the two constructed ANNs trained with reinforcement learning.*

	FFNN	**RNN**
Inputs	v_x, v_y, θ, e_d, ψ_e, v_r	v_x, v_y, θ, e_d, ψ_e, v_r
First layer	15 neurons	15 neurons
Activation	tanh	tanh
Second layer	12 neurons	12 neurons
Activation	tanh	tanh
Third layer	3 neurons (output)	3 neurons (output)

In control applicatios, using integrals and derivatives of signals is common. It could be of use in path tracking as well. To test this, an RNN with the same amount of neurons as the FFNN were created. Its structure is shown in Table 5.3. Since an FFNN does not have the feedback property of the RNN, the new network

structure can offer improvement by allowing these integrals and derivatives to be calculated. The recurrence was implemented such that the input for each hidden layer is the output of the previous layer as well as a time delayed output of the following activation function.

Another difference in the two networks is the number of parameters that have to be set. As shown in Chapter 2, a neuron is a weighted sum and an added bias. When viewing the two networks described in Table 5.3 the number of trainable parameters change due to the increased connections in the RNN. The number of trainable parameters to be set for the FFNN are

$$\sum_{k=1}^{K}(n_{k-1}+1)n_k = 336, \tag{5.1}$$

and for the RNN are

$$n_K \cdot n_{K-1} + \sum_{k=1}^{K-1}(n_{k-1}+1+n_k)n_k = 705, \tag{5.2}$$

where k denotes each layer, n_k the amount of neurons in each layer, n_0 is the input layer and n_K is the output layer. This results in the RNN having more than twice the amount of adjustable parameters compared to the FFNN. As described above the optimization of the RNN is more complex and hence requires more time.

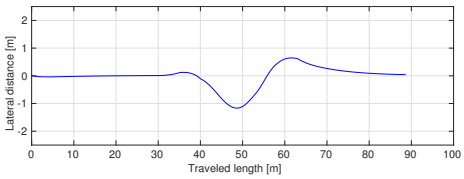
5.3 Results

The performance of the two different types of ANNs created were tested in the Gazebo simulation environment as well as in the custom C++ simulation. The results are presented in this section.

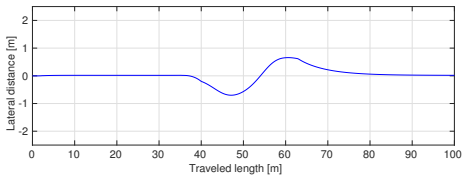
The training results from the C++ simulations using the FFNN and RNN can be seen in Figure 5.2. Both of the controllers manage to follow the paths in these simulations.

By comparing the FFNN and RNN it can be deduced that the RNN performs better in most scenarios. This is the expected result, since the recurrence in the network provides additional information. It also seems to outperform the controllers presented in Chapter 4 in most scenarios. Behaviour wise the two controllers are quite similar. The overall shape of the graphs are almost the same, but the RNN seems to stay closer to the path for the most part. However, the FFNN seems to have learned an interesting strategy. When it approaches a curve, it first steers in the opposite direction.

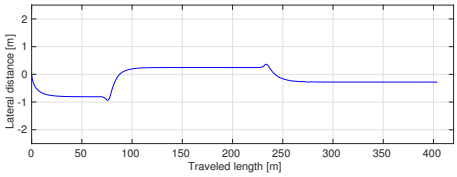
In Figure 5.2b there is a constant error of about 2 cm. Apart from this constant error, the vehicle stays close to the track and behaves well. The easiest solution to get rid of this error would probably be to simply let the RNN train for a longer time. However, there is no guarantee that it would produce a solution without the offset. It might simply be advantageous to stay on the right side of the path on the track used in the training procedure. Extending it with different tracks could potentially also improve the ANN in this regard.



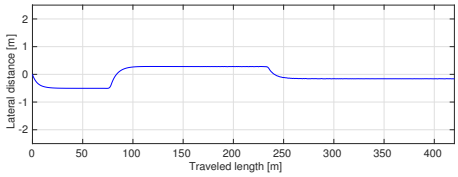
(a) *Lateral distance from the path to the center of the vehicle when following test track A using the FFNN.*



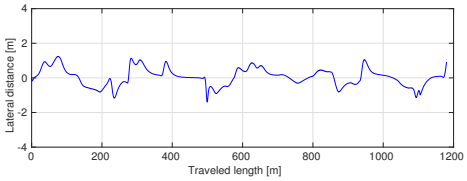
(b) *Lateral distance from the path to the center of the vehicle when following test track A using the RNN.*



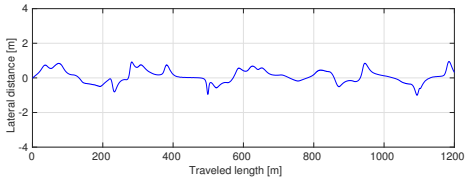
(c) *Lateral distance from the path to the center of the vehicle when following test track B using the FFNN.*



(d) *Lateral distance from the path to the center of the vehicle when following test track B using the RNN.*



(e) *Lateral distance from the path to the center of the vehicle when following test track C using the FFNN.*



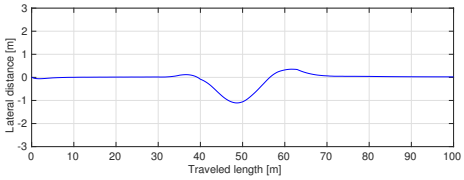
(f) *Lateral distance from the path to the center of the vehicle when following test track C using the RNN.*

Figure 5.2: *Lateral distance from the path to the vehicle when using the FFNN controller and the RNN controller, for each of the test tracks. These simulations where run with the C++ simulation.*

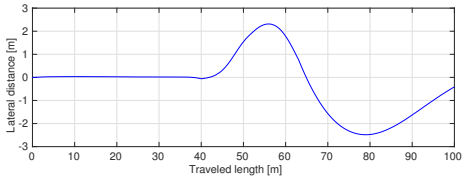
The results from running the Gazebo simulation with the FFNN and RNN controllers are displayed in 5.3. In these simulations the controllers did not manage to keep the vehicle on path for all of the tracks.

Although the controllers showed promising performance in the C++ simulations, the Gazebo simulation did not show as good of a result. The main difference between these simulations were the complexity of the model, therefore this is a likely cause for the poor behaviour. Note especially the distances traveled in Figures 5.3e and 5.3f compared to Figures 5.2e and 5.2f. In the Gazebo simulations, the vehicle was not able to reach the end of the track.

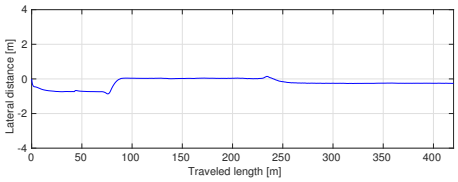
A more in depth comparison between the different controllers and potential solutions to the problems encountered in the Gazebo simulations is presented in Chapter 6.



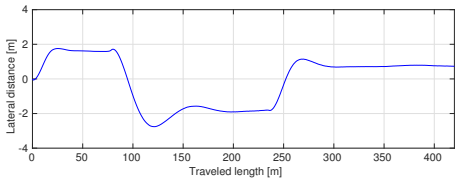
(a) *Lateral distance from the path to the center of the vehicle when following test track A using the FFNN.*



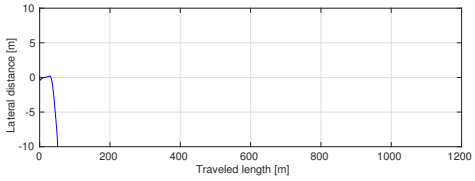
(b) *Lateral distance from the path to the center of the vehicle when following test track A using the RNN.*



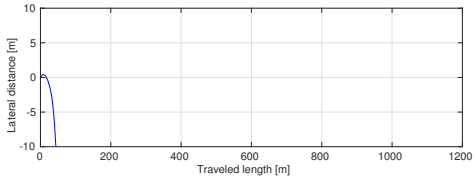
(c) *Lateral distance from the path to the center of the vehicle when following test track B using the FFNN.*



(d) *Lateral distance from the path to the center of the vehicle when following test track B using the RNN.*



(e) *Lateral distance from the path to the center of the vehicle when following test track C using the FFNN.*



(f) *Lateral distance from the path to the center of the vehicle when following test track C using the RNN.*

Figure 5.3: *Lateral distance from the path to the vehicle when using the FFNN controller and the RNN controller, for each of the test tracks. These simulations where run with the Gazebo environment.*

6
Discussion and results

This chapter analyzes the methods used as well as compares the results obtained for each of the methods shown in Chapters 4 and 5.

6.1 Learning methods

Each approach has its obvious advantages and disadvantages. Given that the network to be trained is sufficiently large, a supervised learning method tends to yield a fast result in terms of required training time compared to the method used for reinforcement learning. This is a good way to generate prototype networks and compare different structures and sizes of networks.

A prominent issue with supervised learning methods is the data requirements in order to train the ANN. The quality of the data limits the potential performance of the network and whether it will converge to a controller similar to that which was used for the data generation. Alternatively, the data could be generated by recording a person driving and then used to train a network that imitates how a human would drive a vehicle. But even this may not produce an optimal solution to the problem. Another issue is the sheer amount of data required to represent all scenarios one might face on the road. Because of these data requirements, using supervised learning with this kind of data was not considered a suitable alternative for this thesis.

Using reinforcement learning instead allows the controller to be generated without existing data but impose other problems. To start with, implementing reinforcement learning may be far from simple, and does not guarantee an optimal solution. Also, selecting the evaluation function may be difficult depending on the task at hand, and will greatly impact the behavior of the controller. Essentially the issue in formulating the expression lies in the difficulty in defining what "good driving" is in a mathematical expression. Since there is no formal definition this had to be done without extensive knowledge of the subject. This may affect the controller, since it is trained to minimize the given function, and if that function is lacking in some aspect the resulting controller may be unsatisfactory. Reinforcement learning is however a strong alternative method in order to find a better solution than what the recorded data enables.

Another issue with reinforcement learning methods in control applications is that of model correctness. The goal is typically to create a controller which can be used to steer a mechanical, electrical, or other type of physical system. To enable a computer to learn such a task one needs to let it experiment and explore its

environment. In many cases that might not be safe or practical. As such it is typically carried out in a simulated environment, out of danger. If the simulations are inaccurate, the reinforcement learning algorithm will not provide any realistic results. To some extent this is also true for traditional control methods, which are tested in simulations before being deployed, but tuning such methods are often easier than manually tuning a neural network. The benefit of not having to design a controller by hand is therefore partly reduced by the requirement to use a very exact model.

A weakness of using the GA is a rather slow convergence. Since it is driven purely by randomness, many solution candidates are bad by a large margin. There is also no guarantee that a proposed solution has not already been tested. Other reinforcement learning algorithms relying on "educated guesses" might provide a faster learning rate while yielding comparable results.

The fitness function used in the GA does work in the sense that a working controller is obtained. The question is if an objectively better ANN could have been produced with a different one. The current fitness function is expressed in terms of time. This leads to aggressive behavior, since it is better to do everything fast. This might be a reasonable argument for some of the evaluated system states, e.g. the control error in the velocity. But when measuring the distance to the path this focus on time leads to the controller rushing to get back on the path once it looses it. In a real vehicle, being off the path is associated with danger. The last thing you would want in such a scenario is to increase the throttle. Instead more safety oriented measurements could be included in the fitness function, such as lateral force, to decrease the risk of wheel slippage.

The distance measurements used in the fitness function encourages control of the whole length of the vehicle, which is not usually the case when geometric control methods are used. The control law used in the Stanley method for instance, is good at controlling the position of the front wheels, but the rear wheels are left without consideration. This is a reasonable simplification if a typical car is considered. For larger vehicles such as trucks, this simplification may result in the rear ending up significantly off track. The ability to consider the whole length of the vehicle is a strength with the approach used in this thesis.

6.2 Network structures

An important factor in the ability of the controllers is the structure of the ANNs. The amount of neurons and layers, as well as the used topology all affect its potential effectiveness. Theoretically an RNN should be able to obtain a better performance than that of an FFNN, due to the internal feedback of the network, which can enable important information such as the derivatives and integrals of the system states to be calculated by the ANN. Such information is used in many other control schemes and as such it could be of use in path tracking as well. This thesis has not formally proven that this is the case, but from comparing the results from the different controllers, it can be deduced that the RNN controller has properties not shared with the other controllers that are beneficial in many situations, see Figure 6.1. This is a good thing, since it means that using RNNs opens up actual useful

possibilities.

The structure of the nets, i.e. amount of neurons and their connections, that were used was decided by analyzing the performance of the controllers obtained with supervised learning. Even though the goal was to find a better control strategy, the sizes of these ANNs gave an idea about the minimal size that would be required to match the used controllers, namely the Stanley method and pure pursuit, see Sections 2.2.2 and 2.2.1. Although, a smaller network was considered preferable in order to keep the computational complexity of the controller low. Another benefit of using a smaller network is the number of parameters to be calculated by the GA. Since the behaviour of an ANN depends not only on the individual parameters themselves but rather the combination of parameters, the complexity of the optimization is increased drastically when the number of parameters increase. This is especially noticeable for an RNN, both due to the increased number of weights as well as the internal feedback dynamics of the RNN, which itself enables complex dynamics to be calculated.

This thesis has implemented a GA that changes the weights and biases of an existing network. It would, however, be possible to let the GA adjust other parameters as well, such as number of neurons and layer setup. This could potentially find an optimal amount of neurons, but would most likely require an even longer training time. This could be an interesting area for future work.

6.3 Controller results

A comparison between the performance of the different controllers is shown in Figure 6.1 and 6.2. Figure 6.1 shows the results from the C++ simulations and Figure 6.2 shows the results from the Gazebo simulations.

The results from the C++ simulations shows the potential of the reinforcement learning approach to the path tracking problem. The results from the Gazebo simulations illustrates some of the issues with it. When searching for the best possible solution to a problem, the obtained solution might be very specialized to that particular problem. If the problem is changed, e.g. another model is considered, there is no guarantee that the solution still works.

Although a controller has been generated using these methods there may exist a potential weakness in the robustness of the controller due to that of the ideal sensor assumption, i.e. no noise nor biases in the sensor data. The information used in both the development and evaluation of the controllers was directly gathered from the simulation environment. A more realistic like simulation could be achieved by emulating sensor noise in order to show how well the ANN can handle such a scenario. Doing so would also increase the robustness of the controller. This could potentially solve some of the problems encountered while running the Gazebo simulations with the reinforcement learning controllers.

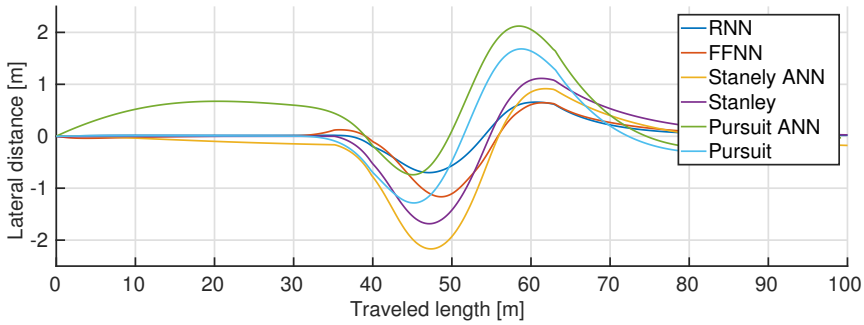
A comparison including all of the control schemes dealt with during this thesis, points to the difficulty of determine which path tracking method to use. The results from the C++ simulations are not consistent with the ones obtained in the Gazebo simulations. The conclusion that can be drawn from this is that there is no single control method which is better in every situation. Each one has their strengths

and weaknesses. Even though a reinforcement learning controller should be able to outperform the geometric path trackers in theory, the model used might be too different from the actual vehicle that a geometric path tracker would perform just as well.

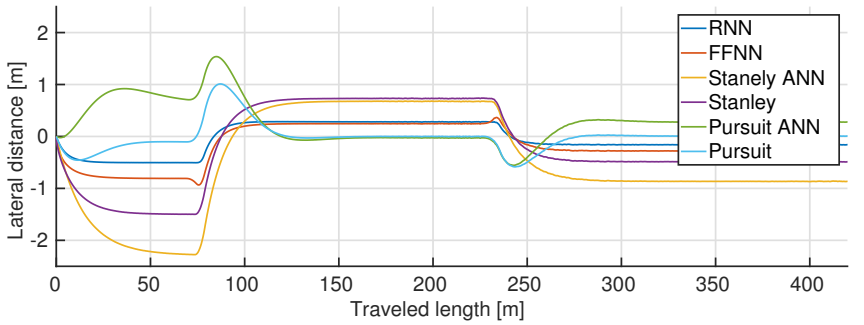
Even though the generated controllers performed well in the C++ simulations, the results from the Gazebo simulations were not satisfactory. The most likely cause for this is the differences in the models used. Even though the FFNN performs better than most other controllers on track A and B, track C reveals some issues with the ANN. The training of the reinforcement learning controllers was done in the C++ simulation, so it is expected that the results is better when using that same simulation. However, the results points to something else. The behavior of the ANNs are unstable. A possible explanation for this is the nature of the GA implementation. When the ANNs get better at their task, they seldom end up far off the track. This means that knowing what to do once that happens becomes unimportant and hence unaccounted for, i.e. the ANN becomes overfitted to the specific task. When the controller is used on a different model where it ends up off track the result is unwanted behaviour. Expanding the training procedure to include a set of different scenarios for the ANN to handle could potentially solve this issue and produce a more stable controller.

Another approach to this problem would be to instead introduce uncertainties in the model by doing the same simulation but with several different model parameter sets. This would in theory force the ANN to be more stable, since the developed controller would have to account for several different possible outcomes of an action, due to an unknown model.

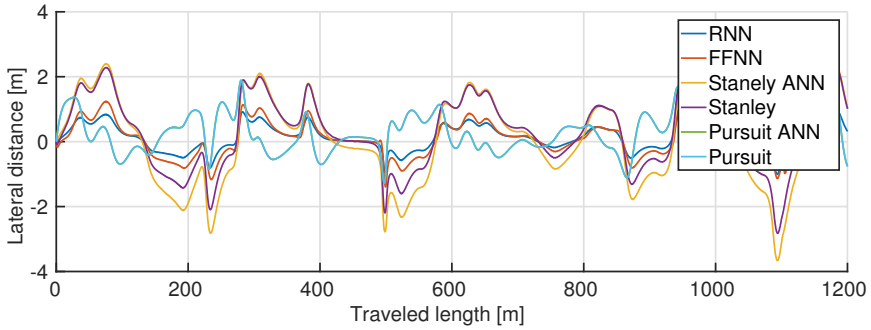
Alternatively, some form of adaptiveness could be added to the control scheme. This would allow the ANN to adjust its parameters to match different vehicles and driving conditions. This could solve the issue of inaccurate models and perhaps even result in a "universal" path tracker that can perform at the same level as the RRN does in the C++ simulations in all scenarios.



(a) *Lateral distance from the path to the center of the vehicle when following test track A.*

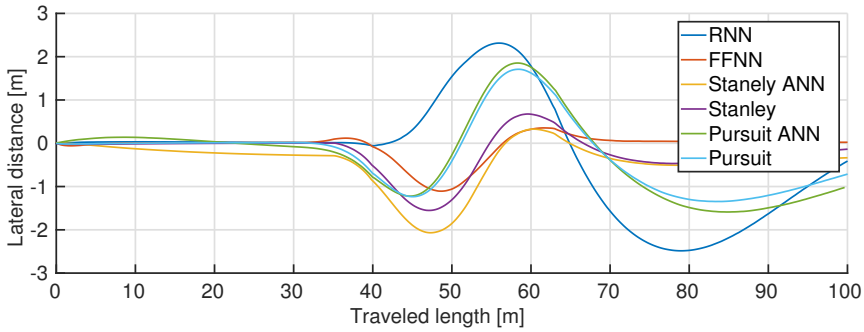


(b) *Lateral distance from the path to the center of the vehicle when following test track B.*

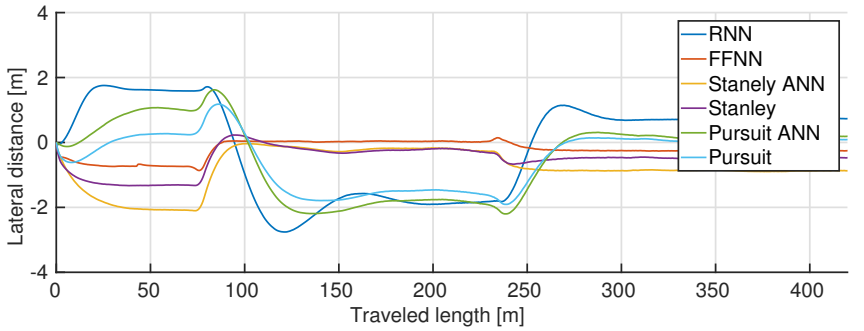


(c) *Lateral distance from the path to the center of the vehicle when following test track C.*

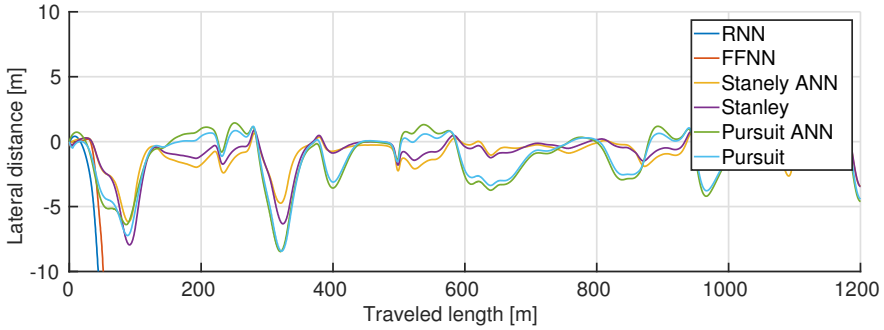
Figure 6.1: *A comparison of the lateral distances in the C++ simulations for the different controllers.*



(a) *Lateral distance from the path to the center of the vehicle when following test track A.*



(b) *Lateral distance from the path to the center of the vehicle when following test track B.*



(c) *Lateral distance from the path to the center of the vehicle when following test track C.*

Figure 6.2: *A comparison of the lateral distances in the Gazebo simulations for the different controllers.*

7
Conclusion and future work

Autonomous vehicles are a highly discussed subject both in today's research as well as within the industry. In order for this to be realized in an actual vehicle there are several parts of the system that have to work together, such as obstacle detection, path planning, path following, control allocation and so on. Another subject that has gained attention in recent years is that of machine learning and neural networks. Through this thesis different neural network path controllers have been developed in order to evaluate whether a neural network is a realistic option to traditional control methods.

7.1 Conclusion

Existing path tracking methods based on static non-linear functions show unsatisfactory results in many cases. Instead, ANNs can be used, which have been proven to be able to imitate any mathematical function. This thesis has shown that ANNs are a good alternative to geometric path trackers. However, the results from the Gazebo simulations points out some problems that have to be addressed before this approach can be used in a physical system. Mainly that the model used for training the ANN affects how well the controller performs. Using a too simple model limits the robustness of the controller once applied in a more realistic environment. Therefore a more realistic training model should improve the results. Additionally, introducing uncertainties during training may also increase the robustness. Another alternative to cope with the model discrepancies would be to make the controller adaptive.

7.2 Future work

Even though the reinforcement learning methods have shown promising results, more work is needed to obtain a controller that can be used in an actual physical system. Specifically, a more sophisticated model which includes tire dynamics needs to be formulated. Additionally, there has not been much work in assuring the safety of the controller. The experiments in the Gazebo simulations shows that this has had a negative impact on the final result, e.g. the ANN has learned to speed up while far from the track to minimize the time away from it. From the authors' point of view, the most important issues that has to be solved for these control schemes to work are:

- Improve the model. The assumption of no wheel slippage encourages dangerous behavior.
- Introduce uncertainties in sensor readings and model parameters. Realistically, no system model or sensor is 100 % accurate. This has to be accounted for if ANNs are to be put in control of autonomous vehicles.
- Investigate different fitness functions. The fitness function used in the GA currently encourages optimallity in the least time sense, i.e. do things as fast as possible. Safety should be of highest priority and this kind of behaviour defies that sentiment.

Bibliography

[1] A. AG. (2018) Audi piloted driving homepage. [Online]. Available: https://www.audi.com/en/innovation/piloteddriving.html

[2] T. M. Corporation. (2018) Toyota concept-i homepage. [Online]. Available: https://www.toyota.com/concept-i/

[3] V. Cars. (2018) Volvo drive me homepage. [Online]. Available: https://www.volvocars.com/intl/about/our-innovation-brands/intellisafe/autonomous-driving/drive-me

[4] M. Maurer, J. C. Gerdes, B. Lenz, H. Winner *et al.*, *Autonomous driving*. Springer, 2016.

[5] J. B. Greenblatt and S. Saxena, "Autonomous taxis could greatly reduce greenhouse-gas emissions of us light-duty vehicles," *Nature Climate Change*, vol. 5, no. 9, p. 860, 2015.

[6] D. J. Fagnant and K. M. Kockelman, "The travel and environmental implications of shared autonomous vehicles, using agent-based model scenarios," *Transportation Research Part C: Emerging Technologies*, vol. 40, pp. 1–13, 2014.

[7] J. M. Snider *et al.*, "Automatic steering methods for autonomous automobile path tracking," *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08*, 2009.

[8] W. G. Baxt, "Application of artificial neural networks to clinical medicine," *The lancet*, vol. 346, no. 8983, pp. 1135–1138, 1995.

[9] B. Ghobadian, H. Rahimi, A. Nikbakht, G. Najafi, and T. Yusaf, "Diesel engine performance and exhaust emission analysis using waste cooking biodiesel fuel with an artificial neural network," *Renewable Energy*, vol. 34, no. 4, pp. 976–982, 2009.

[10] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[11] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

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

[13] Y. Zhang, W. Chan, and N. Jaitly, "Very deep convolutional networks for end-to-end speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*. IEEE, 2017, pp. 4845–4849.

[14] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on neural networks*, vol. 1, no. 1, pp. 4–27, 1990.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, p. 533, 1986.

[16] O. Amidi and C. E. Thorpe, "Integrated mobile robot control," in *Mobile Robots V*, vol. 1388. International Society for Optics and Photonics, 1991, pp. 504–524.

[17] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[18] D. Thomas. (2014) Ros/introduction. [Online]. Available: http://wiki.ros.org/ROS/Introduction

[19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[20] D. Steckelmacher. (2015) nnetcpp. [Online]. Available: https://web.archive.org/web/20180518104617/https://github.com/steckdenis/nnetcpp

[21] M. Wall. (1999) Matthew's galib: A c++ genetic algorithm library. [Online]. Available: http://lancet.mit.edu/ga/

[22] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *Intelligent Vehicles Symposium (IV), 2015 IEEE*. IEEE, 2015, pp. 1094–1099.