Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Fachgebiet Regelungstechnik

Pontificia Universidad Católica del Perú
Escuela de Posgrado

Autonomous Control of a Mobile Robot with Incremental Deep Learning Neural Networks

Master's thesis in fulfilment of the requirements for the degree of

Master of Science (M.Sc.)

in Technische Kybernetik und Systemtheorie

and

Magíster

en Ingeniería de Control y Automatización

Isabella Glöde

Study Program:	International Double Degree Dr.
Supervisor (PUCP -Lima):	Antonio Manuel Morán Cárdenas
Supervisor (Co-asesor TU -Ilmenau):	Prof. Dr.-Ing. Johann Reger

This thesis was submitted on February 25th, 2021.

ET LUX IN TENEBRIS LUCET
MCMXVII

Abstract

Over the last few years autonomous driving had an increasingly strong impact on the automotive industry. This created an increased need for artificial intelligence algorithms which allow for computers to make human-like decisions. However, a compromise between the computational power drawn by these algorithms and their subsequent performance must be found to fulfil production requirements.

In this thesis incremental deep learning strategies are used for the control of a mobile robot such as a four wheel steering vehicle. This strategy is similar to the human approach of learning. In many small steps the vehicle learns to achieve a specific goal. The usage of incremental training leads to growing knowledge-base within the system. It also provides the opportunity to use older training achievements to improve the system, when more training data is available.

To demonstrate the capabilities of such an algorithm, two different models have been formulated. First, a more simple model with counter wheel steering, and second, a more complex, nonlinear model with independent steering. These two models are trained incrementally to follow different types of trajectories. Therefore an algorithm was established to generate useful initial points. The incremental steps allow the robot to be positioned further and further away from the desired trajectory in the environment.

Afterwards, the effects of different trajectory types on model behaviour are investigated by over one thousand simulation runs. To do this, path planning for straight lines and circles are introduced. This work demonstrates that even simulations with simple network structures can have high performance.

ET LUX IN TENEBRIS LUCET
MCMXVII

Contents

ET LUX IN TENEBRIS LUCET
MCMXVII

1 Introduction and Motivation

1.1 Introduction

Since the 1960s, there has been a large interest in understanding machine learning and, more specifically, the use of neural networks to achieve it. However, given the limitations of computational power at the time, it would take another 20 years, until the 1980s, for our more modern interpretation of neural networks to be established. Now, given their potential applications in a variety of fields, research into neural networks have garnered massive interest [1].

Nowadays, a lot of deep learning strategies are implemented in almost every type of system including optimization systems, diagnostic systems, and control systems. Starting with industry 4.0, many companies are starting to save huge data sets and need new strategies to analyse them. For example, services which require the analysis of such data sets, like Spotify, could not exist without such algorithms. Now, in order to overcome challenges facing their automated driving assistants and warning systems, such as the navigation of unknown territories, parking in tight spaces, or the navigation of narrow streets, the automotive industry has invested heavily in development of these algorithms.

The transition to technical supported solutions offers facilitation of the task but also leads to a longer development process in advance. The application of neural networks is nowhere more so apparent than (uncrewed) vehicles. Utilizing the sensors available to it, the vehicle must gather information about its environment, plan a course of action, and execute the plan to achieve a specific goal. Currently, these systems are controlled remotely, however systems which could be completely automated or even work in tandem with other automated machines or vehicles would be revolutionary.

The focus of this work is to utilize deep learning applications to navigate a mobile four-wheel steering (4WS) robot on different target trajectories. This can later be

used to navigate (uncrewed) vehicles within dangerous terrain for remote monitoring or surveillance applications.

1.2 Motivation

Unmanned reconnaissance and defense systems are increasingly attracting attention. The advantage of free actions while being in a difficult situation leaves the interest in many different sectors of public life. The majority of those systems are aircrafts, often used for military purposes. But also land and water vehicles appeal. Unmanned underwater vehicles are used for reconnaissance and mapping. Often there are used swarms of robots to 3D map simultaneously the ground. The land vehicles have their application fields more in reconnaissance causes.

The ability of unmanned reconnaissance and defense systems to perform logical decisions to accomplish specific tasks autonomously has been demonstrated by numerous applications within the defense and aerospace sectors. Further, the ability of such systems to communicate in networks allows, for example, 3D mapping of the ocean floor. These commercial successes have attracted both public and scientific attention. The use of land vehicles is of particular interest given the unique challenge of operating upon a variety of different surfaces. The specifications of the land vehicle's steering system play a critical role in the exact directions the vehicle can move, and therefore greatly impact the algorithms that can be used to govern a vehicle's movement. 4WS systems provide advantages over their modern counterparts due to:

- their reduced turning radius— due to the ability of the front and rear wheels to steer simultaneously,
- their increased agility at low and medium speeds— due the ability of the front and rear wheels to steer in opposite,
- their increased stability at higher speeds— due to the ability of the front and rear wheels to steer in the same direction.

Due to a better maneuverability and high-speed stability the automotive industry applies the system in their new cars. The aforementioned advantages of such systems have been implemented into new vehicles for small steering angles such as the Acura TLX, Audi A8, BMW 7 Series, and Cadillac CT6. More exaggerated versions of these

systems are often implemented into special-purpose vehicles. Fire engines, for example, exhibit such forms of pronounced steering (as observed in 1.1). This allows for such a large vehicle to retain manageable maneuverability so it can provide assistance in very tight or small spaces.

Figure 1.1: Fire truck with 4WS [2]

In reconnaissance vehicles 4WS technology is being paired to a control system which is tasked with dealing with nonlinear system dynamics. In many situations there is no clear specification of the setting. Therefore it is desirable to let the system make a decision based on the present setting.

This decision can be made statically with simple queries. Another approach is to use learning strategies to make the decisions more dynamic. The dynamization of the decisions helps to control the nonlinear system. Advantages of this approach are the easier implementation of those algorithms and having the opportunity to enlarge the skill set of the controller. Also there are great benefits for growing data sets and as a consequence growing knowledge of the system.

1.3 Objective of the work

The objective of this work is to use incremental deep learning to train and validate a 4WS robot for autonomous positioning and tracking under different scenarios.

Several previous works have dealt with modelling and parameter estimation of those systems. Because of that, the main focus in the proposed master thesis will be in following a target trajectory using learning algorithms. Part of the validation will be, to change parameters of the network as well as those of the robot and the trajectories for different driving scenarios. To accomplish this objective the following topics will be deeply examined in this work:

- reproduction and validation of an exact but simple mathematical model of an 4WS vehicle,
- design of an adapted neuro controller,
  - tuning of the hyperparameters of neuro controller,
  - implementation of the learning strategy,
- pathplanning,
- validation with different scenarios,
- comparison of the behaviour of different trained networks,
- simulating system, controller and environment to portray system, controller and environment.

Since the main aspect of this work is to simulate the system, MATLAB® was used to implement and validate different algorithms. The goal is to create a complete software package that can challenge the trained robot with different scenarios. Therefore computational resources are needed. The presentation of the results and the illustrations will be realized with LaTeX.

1.4 Outline and content

The objective of this work is to evaluate incremental deep learning with the model of a 4WS mobile robot. Therefore the developed software must be able to change hyperparameters and different control scenarios.

First the state of the art is presented, including different control strategies for 4WS vehicles. Approaches like H_∞ as well as fuzzy control are considered.

Secondly the mathematical model of a 4WS robot is introduced. This includes the derivation of the equations and the validation of the model. Therefore a software piece is presented to display the output.
Then a neuro controller is designed to calculate the input of the system to reach a certain point or follow a trajectory. This neuro controller is parameterized and trained with different tasks and scenarios with an incremental deep learning strategy.

In the end a standard parameter set is used to simulate various cases including a examination of different parameters. Ultimately, the results will be critically reflected upon.

The content presented in this thesis is organized as follows:

- *Chapter 2* presents the state of the art. Including existing control strategies for 4WS vehicles and application fields of incremental learning strategies.
- *Chapter 3* describes the model of a general 4WS vehicle. The development of the mathematical model is explained as well as an simulated validation.
- *Chapter 4* contains an overview over deep learning neural networks. Learning strategies and the training process are described. Also the control problem is defined.
- *Chapter 5* introduces the implementation of the system with its architecture and used algorithms.
- *Chapter 6* presents the simulation setup as well as the resulting data.
- *Chapter 7* concludes the thesis with a summary of the main results and discusses the outlook for possible future topics of research.

ET LUX IN TENEBRIS LUCET
MCMXVII

2 State of the Art

This chapter provides an overview over the latest developments in the field of 4WS vehicles, applications with incremental deep learning (IDL), as well as the design of 4WS models using the SUMMIT example. It holds the information used in further chapters. The existing works are presented in the following sections.

2.1 Model design of four wheel steering vehicles

2.1.1 Model of the mobile robot SUMMIT

In previous works with the mobile robot SUMMIT a nonlinear one track model of SUMMIT was designed, parametrized and validated.

In 2016, the TU Ilmenau introduced a project to test autonomous driving using the mobile robot SUMMIT [3]. Part of this project was to obtain a valid nonlinear system of the robot, to implement a laser scanner for obstacle avoidance and to deploy a real time control system.

The first work was written in 2013 [4]. It includes a one track model with consideration of influences of tires as well as first considerations to model predictive control (MPC). All of the further works [5, 6, 7, 8] consider the same one track model but with different numbers of considered states in the system. They differ in the determination of the model parameters.

The first, [5], provides a mechanical discussion about parameters like inertia. The second, [7], utilizes the findings of [9] to optimize model parameters using neural network (NN) algorithms. [8] then utilizes a NN to estimate the real steering angle out of a calculated angle. These calculations offer increased accuracy which is further validated in [8] due to the ability of NN to intercept and evaluate disturbances and sensor noise.

Neural networks are very well suited to intercept and evaluate disturbances and sensor noise.

2.1.2 Model of a four wheel steering vehicle

Usually a two degree of freedom (DOF) dynamics model, the bicycle model, is used. The advantage is in the handling ability. The model consists of two states: the sideslip angle (lateral movement) and the yaw rate.

This model is consulted by [10, 11, 12, 13, 14, 15]. In [16] a lateral dynamics model for four wheel steering vehicles is described. It also belongs to the two DOF models. It does not include roll and pitch movement, as well as the impact of the steering mechanism itself.

The three DOF model is described in [17]. In addition to lateral movement and yaw rate it considers roll motion. Thereby the accuracy of the model increases.

[11] introduces a seven DOF model, where roll, pitch and vertical dynamics are considered. It is validated and used in the paper.

Another approach is to use a nonlinear model to train a radial basis function (RBF) neural network. The results are described in [18].

2.2 Control strategies for four wheel steering vehicles

The control of a 4WS vehicle is a demanding task because of its nonlinear dynamics. Often there are two parts to be implemented for the control. The first part reduces the influence of model uncertainties and the second part is the actual control algorithm. The broad variety is described in the following paragraphs.

2.2.1 Model predictive control

As mentioned in 2.1, MPC is often discussed for SUMMIT. The topic was reviewed by further works from [4, 5, 6]. All of them are using MPC to follow trajectories. [8] uses a Kalman filter for an increase of robustness in the prediction phase of the system. It extends the model predictive controller and gives the possibility to tune the

robustness.
It is easy to see that the project is using MPC and developed this approach further and further. While in the beginning only model parameters were optimized, in the end dynamic approaches were implemented, which were supposed to intercept the discrepancies between model and reality.
In contrast to the previous work, this master's thesis looks to achieve these goals using learning algorithms.

2.2.2 Optimal control

The papers of [16, 10] are using an H_∞ optimum controller as a control strategy. Simulations in [16] prove the applicability of an H_∞ controller for an ideal steering model. The transient response can be improved in contrast to a front wheel steering vehicle. On the other hand, [10] uses an H_2 optimal controller and a Kalman filter for the ideal model. Success is achieved at low speeds. Because of model uncertainty, an H_∞ controller is designed [10] and shows robustness, adaptability and stability.

2.2.3 Sliding mode control

The research of [11, 19, 14, 15, 20] delineate a sliding mode control strategy. The objective is to improve vehicle handling stability. Input variables are rear wheel steering and yaw moment. The overall achievement is an improved robustness. With [11] this is accomplished by the use of the center of gravity slip angle and yaw rate. The measurement of those parameters is not applied in many systems.

[19] introduces an approach for discrete time sliding mode control. It shows that a continuous controller and the proposed discrete controller do have similar performance.

2.2.4 Fuzzy control

Fuzzy control is not based on a mathematical model of the system. It is based on human knowledge of the system and has predefined rules. But to obtain data, often a simple model is used [17, 13].

The simple fuzzy control approach is used in [17]. By using a three DOF model of a 4WS vehicle the fuzzy rules are applied. The controller inputs are velocity and front wheel angle and the output is the rear wheel angle. With the 3 DOF model sideslip

angle, yaw rate, roll angle and roll angular velocity are generated.
Also [13] deals with fuzzy controllers. Here, excessive understeering at high speeds is the challenge. As an input, yaw error and side slip angle error, as well as their respective derivatives are selected. Output of the system is the direct yaw moment. The system is parted in the side slip angle fuzzy logic controller and the yaw fuzzy logic controller.

In [12] a fuzzy NN system is designed. The NN is introduced as a computing tool and generates reasonable fuzzy rules through learning. Lastly the fuzzy system will be optimized by a genetic algorithm. The fuzzy NN controller has five layers and generates the steering angle of the rear wheel, which is the input of an NN, which calculates yaw rate and side slip angle. A similar approach is chosen in [21].

2.2.5 Neural network control

Starting from fuzzy NN, it leads to the pure neural networks approaches. Although the NN does not need an exact mathematical model, training data can be obtained through simple models, to proof the performance on the real plant afterwards.

Neural networks can be classified in two types: static and dynamic networks. Static networks are networks where the outputs are generated at the same time when the inputs are introduced in. There is not a temporal relationship between inputs and outputs, nor a dynamic relationship between them. But with applications in robotics there is always a relationship within the time. So dynamic networks define that the present outputs do not depend on external inputs only, but also on past network outputs or past network states. Thus the network has a dynamic and memory. Those networks are used for, among other things, control problems, filtering, language processing, and patter recognition.

For the training of dynamic neural networks two algorithms are used: Back Propagation Through Time (BPTT) and Dynamic Back Propagation (DBP). The BPTT Algorithm was proposed by Paul Werbos [22]. It is basically the application of the standard error back propagation algorithm to networks that are augmented by unfolding the original network at each time step.
The DBP Algorithm was proposed by Kumpati Narendra [9, 23]. It is an efficient way to compute the total partial derivatives of the dynamic network outputs with re-

spect to weighting coefficients by using a recursive equation integrating present simple derivatives and previous-step total partial derivatives.
More information about neural networks and their algorithms will be given in chapter 4.

To imitate a human driver a feedforward neural network is proposed in [24]. A feed forward NN is a nonrecurrent network, so it has no feedback connection. The control system consists of two parts, each with a two layered NN. The first part acts as an emulator and the second represents a feedforward controller. The simulations include side winds and obstacle avoidance maneuvers.

In addition to a NN controller a model based on a NN is used in [18]. The structure of the NN controller is synthesized with an radial basis function model. The inputs of this systems are yaw velocity, steering angle of front wheel and the reciprocal of the velocity. The output is as usual the steering angle of the rear wheel.

The NN direct inverse control is represented [25]. It consists of two parts. First, the output of the inverse model is equal to the input of the controlled object. Then, if the error is nearly zero, it can be applied as a controller. The NN has one hidden layer with 12 neurons. Inputs are the steering angle of the front wheel and the slip angle.

2.3 Application fields of incremental learning strategies

Deep learning methods are powerful approaches with a large field of applications. Incremental deep learning as a strategy is likely used for problems with high complexity but also for problems with a growing data set. The idea is to do the learning process like human kind in small steps. It is based on the learning process of babies. So one step itself does not conclude the objective, but leads to a bigger task.

One often mentioned application is to use incremental deep learning for face recognition. It is considered in [26, 27, 28]. With this approach it is possible to recognize a face from different angles. Also there is the option of two different networks: one for facial features and the other for full face detection.

Not only face recognition is an often used application also object recognition [29], speech recognition, natural language processing and computer vision [30] are common focus areas.

Skill learning like the imitation of human behaviour is described in [31, 32].

Another application is the learning of classifiers [33]. That means that reinforcement learning is used to learn a policy to incrementally build neural network classifiers. Reinforcement learning in combination with incremental learning can be found in [34] for the adjustment to dynamic environments on the basis of maze navigation in a warehouse. Also [35, 36] deal with incremental learning in nonstationary environments.

As mentioned in section 2.1.2 neural networks are used for model design as well. The real time modelling of robot dynamics using incremental learning is depicted in [37].

3 Modelling and Computation

Models of technical systems are only approximations of reality. The parameters used to approximate this reality can greatly vary in their complexity. As such, finding the correct parameters which are both as simple as possible and maintain the desired accuracy of the model represents a key challenge when building such a system. In the formulation of models of systems the focus is identifying the dynamical relationship between inputs and outputs. These relationships are used in the controller design process. There are several assumptions made in this work to describe a 4WS robot with its complex, nonlinear, underactuated and nonholonomic system.

3.1 Assumptions

The general 4WS vehicle has a rectangle shaped body with a length and a width. Most of the models are based on low speed, so that side slipping can be neglected, as it only occurs to a minor extent. Likewise is 3D-behaviour not part of the mathematical model. That means, that the motion of the vehicle is determined by geometrical considerations. Masses, inertias and frictional forces are omitted. The acceleration of the vehicle is considered as zero, so the vehicle has a constant velocity.

Due to the ability of the robot to steer both its front and rear wheels, its behavior remains the same when moving forward or backwards. For the control, it is assumed that full state information is instantly available.

Ultimately, the following assumptions are made:

- Only 2D movement in the x-y-plane is being considered.
- Low speed so that dynamical effects are neglected.
- No longitudinal or lateral slip is considered.
- Wheels are not modeled, and friction is neglected.

3.2 State description

Considering the 4WS vehicle in the 2D-plane, the coordinates (x_1, y_1) describe the middle of the front axis of the robot. The angle Ψ describes the orientation of the robot. In positive x direction the angle will be zero and will rise counterclockwise as one can see in figure 3.1.

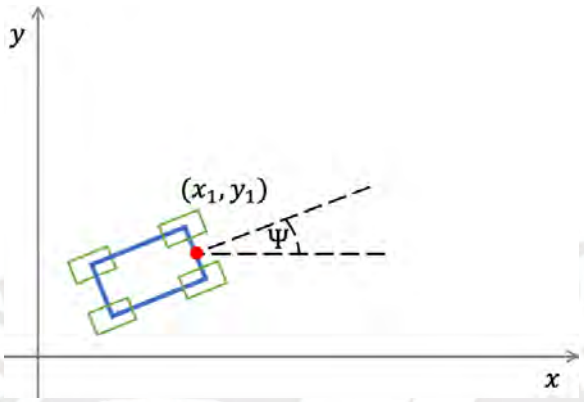


Figure 3.1: State of the robot

The 4WS vehicle is controlled by the steering angles of the front axis and the rear axis. Therefore two steering angles δ_F and δ_R are introduced. Both are limited with

$$\delta_{min} \leq \delta_F \leq \delta_{max}, \ \delta_{min} \leq \delta_R \leq \delta_{max}. \tag{3.1}$$

To simplify the model it is assumed that $-\delta_{min} = \delta_{max}$.

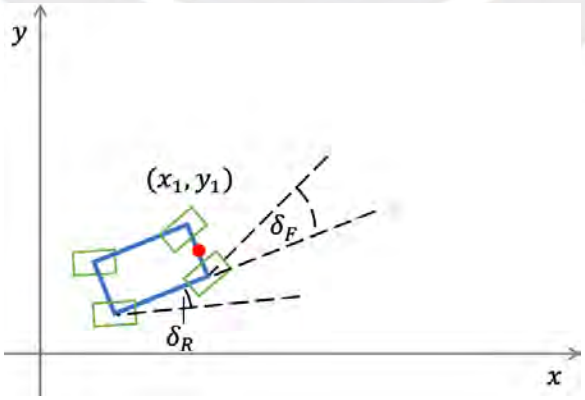


Figure 3.2: Steering angles of the robot

To complete the description of the robot, there are length and width of the robot. In

the following a single track model will be introduced. Therefore the width of the robot will be irrelevant. Only for visualization causes a width will be used.

3.3 Identification of vehicle dynamics

The model of a 4WS robot consists of a function which determines the new state by using the current state and the given input

$$\dot{x} = f(x, u), \tag{3.2}$$

with $x \in \mathrm{R}^n$, $u \in \mathrm{R}^m$ and $f : \mathrm{R}^{n+m} \rightarrow \mathrm{R}^n$.

To obtain a mathematical model the system equations of the mobile robot SUMMIT will be used. The single-track model can be taken from the works of [8] and [5]. However, in order to make the model more interesting and variable, the formulation will be generalized. In the following not only steering in the opposite direction (counter steering), but also front wheel steering, diagonal steering, back wheel steering and independent steering (the front axis is fully independent from the rear axis) are analysed.

In [4] and [5] a mathematical model with five states was established. But if the robot has a low velocity the model can be simplified to a three-state-model. So the state and control variables are defined as

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_{pos} \\ y_{pos} \\ \Psi \end{bmatrix}, \quad u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \delta_{front} \\ \delta_{rear} \end{bmatrix} \tag{3.3}$$

with x_{pos} and y_{pos} as the position of the robot in a x-y-plane, Ψ as the yaw angle, v as the velocity of the robot in meters per second and δ_{front}, δ_{rear} as the steering angle for the front axle and the rear axle. In the following x is used to describe the position of the robot in x-direction, in order to reduce the number of indices. In figure 3.3 a detailed version of the model can be observed. It will be the foundation of the following equations.

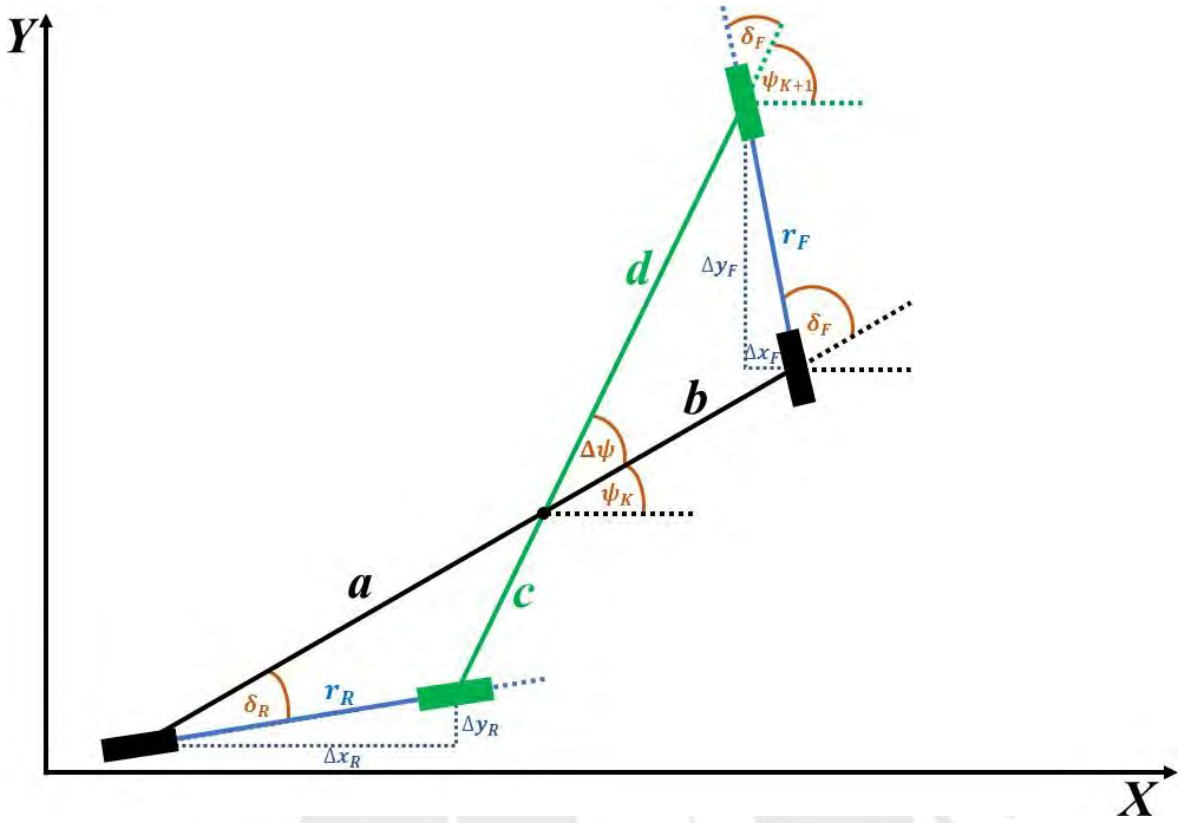


Figure 3.3: Movement of robot in time interval $[k, k+1]$

First part of the consideration is the distance how much each wheel moves in one time step. The rear wheel moves a distance r_R and the front wheel a distance r_F . This can be observed in figure 3.4. Thus the traveled distance in the x-y-plane can be determined as follows:

$$\Delta x_F = r_F \cos(\Psi_k + \delta_F), \tag{3.4a}$$

$$\Delta y_F = r_F \sin(\Psi_k + \delta_F), \tag{3.4b}$$

$$\Delta x_R = r_R \cos(\Psi_k - \delta_R), \tag{3.4c}$$

$$\Delta y_R = r_R \sin(\Psi_k + \delta_R), \tag{3.4d}$$

with $\Delta x = x_{k+1} - x_k$. The same applies to y and Ψ. After this step [8] simplifies the equations, but since the model is supposed to be generalized, they are not introduced. In order to display the global robot movement Δx and Δy the following equation has

to be used

$$\Delta x = \frac{\Delta x_F + \Delta x_R}{2}. \tag{3.5}$$

Using 3.4 and 3.5 follows:

$$\Delta x = \frac{r_F \cos(\Psi + \delta_F) + r_R \cos(\Psi - \delta_R)}{2}, \tag{3.6a}$$

$$\Delta y = \frac{r_F \sin(\Psi + \delta_F) + r_R \sin(\Psi - \delta_R)}{2}. \tag{3.6b}$$

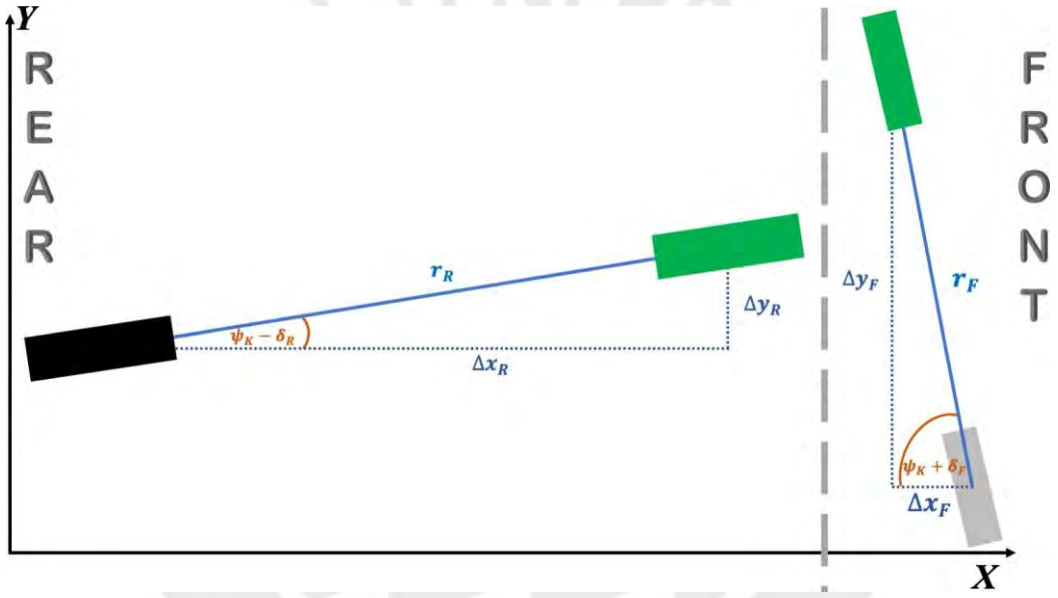


Figure 3.4: Close-up of front wheel and rear wheel in time interval $[k, k+1]$

By using the definition of the derivative and respective limit one can obtain the time derivative of x and y:

$$\dot{x} = v\frac{\cos(\Psi + \delta_F) + \cos(\Psi - \delta_R)}{2} \tag{3.7a}$$

$$\dot{y} = v\frac{\sin(\Psi + \delta_F) + \sin(\Psi - \delta_R)}{2}. \tag{3.7b}$$

Second part of the consideration is the yaw angle Ψ, which also changes. This can be observed in figure 3.3. Now the marked distances a, b, c and d are used. In figure 3.5 the front part of the vehicle is displayed. The objective is to find a connection between

the lengths b and d and $\Delta\Psi$. In addition the height of the triangle is marked with h_F . Considering now a small time interval, Δt, the equations 3.6 can be transformed into Using the general trigonometric functions follows:

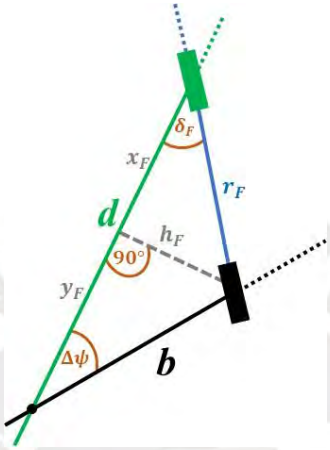


Figure 3.5: Close-up of front wheel in time interval $[k, k+1]$

$$\begin{aligned} d &= x_F + y_F \\ \cos(\delta_F) &= \frac{x_F}{r_F} \\ \cos(\Delta\Psi) &= \frac{y_F}{b} \end{aligned} \tag{3.8}$$

$$\Rightarrow d = r_F \cos(\delta_F) + b\cos(\Delta\Psi). \tag{3.9}$$

Using the sine leads to:

$$\begin{aligned} \sin(\delta_F) &= \frac{h_F}{r_F} \\ \sin(\Delta\Psi) &= \frac{h_F}{b} \end{aligned} \tag{3.10}$$

$$\Rightarrow r_F \sin(\delta_F) = b\sin(\Delta\Psi). \tag{3.11}$$

The same equations can be formulated for the rear wheel axle.

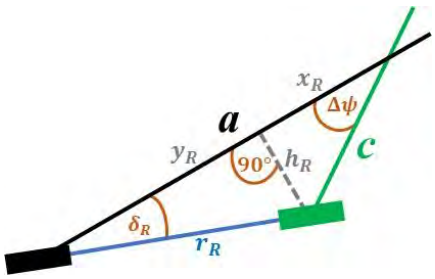


Figure 3.6: Close-up of rear wheel in time interval $[k, k+1]$

This leads to:

$$\begin{aligned} a &= x_R + y_R, \\ \cos(\delta_F) &= \frac{y_R}{r_R}, \\ \cos(\Delta\Psi) &= \frac{x_R}{c}, \end{aligned} \tag{3.12}$$

$$\Rightarrow a = c\cos(\Delta\Psi) + r_R\cos(\delta_F), \tag{3.13}$$

$$\begin{aligned} \sin(\delta_F) &= \frac{h_R}{r_R}, \\ \sin(\Delta\Psi) &= \frac{h_R}{c}, \end{aligned} \tag{3.14}$$

$$\Rightarrow r_R\sin(\delta_R) = c\sin(\Delta\Psi). \tag{3.15}$$

Considering that $\Delta\Psi$ is relatively small and using equations 3.13 and 3.9, leads to:

$$\begin{aligned} a + d &= c\cos(\Delta\Psi) + r_R\cos(\delta_R) + r_F\cos(\delta_F) + b\cos(\Delta\Psi) \\ &= b + c + r_R\cos(\delta_R) + r_F\cos(\delta_F). \end{aligned} \tag{3.16}$$

Furthermore a function $f(L, r_F, r_R, \delta_F, \delta_R)$ has to be found, which describes $b + c$ or $a + d$ with known or controllable parameters. Therefore the equation $2L = a + b + c + d$ is used. L describes the length of the mobile robot. Accordingly follows

$$\begin{aligned} a + d &= 2L - b - c = b + c + r_R\cos(\delta_R) + r_F\cos(\delta_F), \\ 2b + 2c &= 2L - (r_R\cos(\delta_R) + r_F\cos(\delta_F)), \\ b + c &= L - \frac{1}{2}(r_R\cos(\delta_R) + r_F\cos(\delta_F)). \end{aligned} \tag{3.17}$$

By using the equations 3.11 and 3.15 follows:

$$(b + c)\sin(\Delta\Psi) = r_R \sin(\delta_R) + r_F \sin(\delta_F). \tag{3.18}$$

$\sin(\Delta\Psi)$ can be approximated with $\Delta\Psi$. Also it can be observed, in figure 3.3, that the traveled distances by the wheels are the same $r = r_F = r_R$. By equating the equations 3.18 and 3.17 results

$$L - \frac{r}{2}(\cos(\delta_F) - \cos(\delta_R)) = \frac{r(\sin(\delta_F) + \sin(\delta_R))}{\Delta\Psi}. \tag{3.19}$$

For small angles δ_F and δ_R the cosine term can be omitted

$$L\Delta\Psi = r(\sin(\delta_F) + \sin(\delta_R)). \tag{3.20}$$

Considering now a small time interval Δt and with $\lim_{\Delta t \to \infty} \Delta\Psi = \dot{\Psi}$ the equation 3.21 can be transformed into

$$\dot{\Psi} = \frac{v}{L}(\sin(\delta_F) + \sin(\delta_R)). \tag{3.21}$$

Finally the dynamics of the mobile robot summit are governed by

$$\Sigma_{robot}: \begin{matrix} \dot{x} = v\frac{\cos(\Psi+\delta_F)+\cos(\Psi-\delta_R)}{2} \\ \dot{y} = v\frac{\sin(\Psi+\delta_F)+\sin(\Psi-\delta_R)}{2} \\ \dot{\Psi} = \frac{v}{L}(\sin(\delta_F) + \sin(\delta_R)) \end{matrix} \tag{3.22}$$

The length L of the robot is given in [8] with $L = 0.37$ meters.

Additional theorems of sine and cosine

For the circular functions $\sin(t)$ and $\cos(t)$ the following relations apply

$$\cos(\alpha \pm \beta) = \cos\alpha\cos\beta \mp \sin\alpha\sin\beta, \tag{3.23a}$$

$$\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \sin\beta\cos\alpha. \tag{3.23b}$$

Counter wheel steering

By using the addition theorems and the equality for counter wheel steering $\delta_F = \delta_R = \delta$ the system equation will change:

$$\begin{aligned}\dot{x} &= v\frac{\cos(\Psi + \delta) + \cos(\Psi - \delta)}{2} \\ &= v\frac{1}{2}(\cos(\Psi)\cos(\delta) - \sin(\Psi)\sin(\delta) + \cos(\Psi)\cos(\delta) + \sin(\Psi)\sin(\delta)) \\ &= v\cos(\Psi)\cos(\delta)\end{aligned} \tag{3.24a}$$

$$\begin{aligned}\dot{y} &= v\frac{\sin(\Psi + \delta) + \sin(\Psi - \delta)}{2} \\ &= v\frac{1}{2}(\sin(\Psi)\cos(\delta) + \sin(\delta)\cos(Psi) + \sin(\Psi)\cos(\delta) - \sin(\delta)\cos(Psi)) \\ &= v\sin(\Psi)\cos(\delta)\end{aligned} \tag{3.24b}$$

$$\Sigma_{robot_{counterwheel}} : \begin{cases} \dot{x} = v\cos(\Psi)\cos(\delta_R) \\ \dot{y} = v\sin(\Psi)\cos(\delta) \\ \dot{\Psi} = \frac{2v}{L}(\sin(\delta)) \end{cases} \tag{3.25}$$

The resulting system can be compared with [8].

Controlling with a reduced state model

For the control of the given system, it can be useful to reduce the state representation. While it is possible to describe a model with more states than needed, a reduced state with $x \in \mathbb{R}^2$ is introduced. The model with its three states and two inputs is under-actuated. It can be transformed in an actuated system when using the reduced state with two components and an input also with two components. This can improve the training process of the NN, as fewer inputs simplify the training. By testing different configurations it was found that two inputs suffice to reach all of the target types. This might be to the fact that the missing value can be obtained with the state and the target.

3.4 Model validation

Since this work has no real-life object available, the validation is implemented with MATLAB®. In four different scenarios the possible ways to use the model are simulated. Therefore counter wheel steering, front wheel steering and diagonal steering will be tested. The scenarios will be explained and validated with the help of figures. These figures display the robot in the x-y-plane with its wheels and their turnaround. The counter wheel steering results, from figure 3.9, can be compared with those from [8].

3.4.1 Straight ahead

The first scenario is to drive straight ahead. Therefore the steering angles will be zero. In reality, there will be disturbances for the robot. Here it is assumed that the robot moves along a straight line. The straight lines in figure 3.7 have an angle of $\Psi = 0$ (left) and $\Psi = \frac{\pi}{4}$ (right) to the x-axis. 300 steps were simulated with a time step of 0.01 seconds and a velocity of $3\frac{m}{s}$.

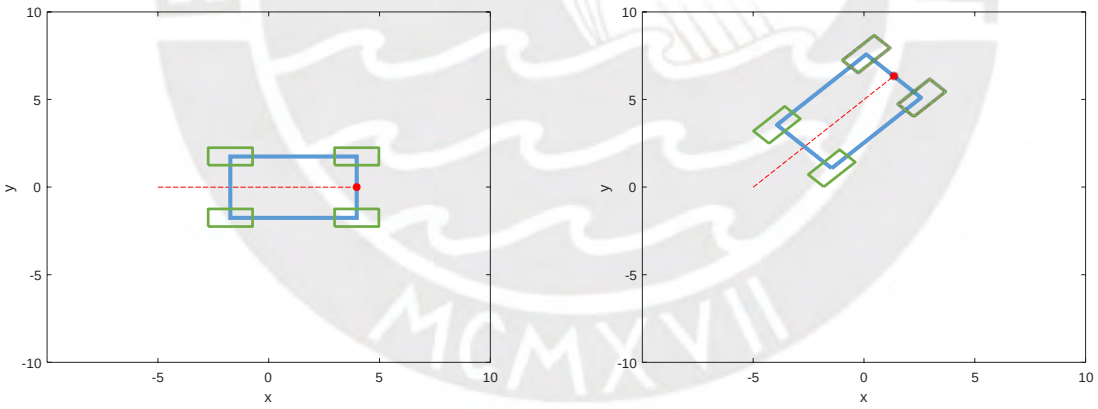


Figure 3.7: Straight ahead Drive with $\delta_F = \delta_R = 0$

In figures 3.7 the movement of the point lying in the middle of the front axis can be seen. It is like expected a straight line. The wheels are not steered.

3.4.2 Front wheel steering

The advantage of the mobile robot is, that all kinds of steering are possible. How the model behaves with only front wheel steering will be shown in this section. The difference to counter wheel steering will be noticeable in driving through curves. That is why the simulation works with a constant front wheel steering angle $\delta_F = \frac{\pi}{4}$. The curve diameters can be compared to counter wheel steering in the following section.

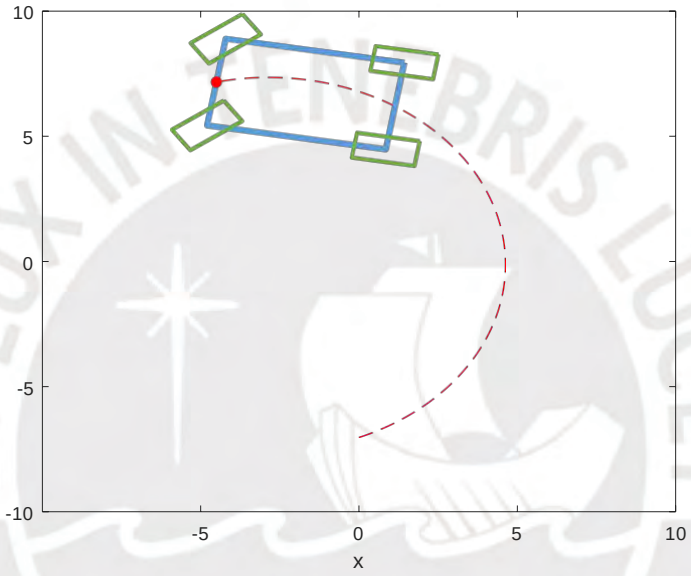


Figure 3.8: Front wheel steering with $\Psi_{init} = 0$ and $\delta_F = \frac{\pi}{4}$ and $\delta_R = 0$

Finally, a large semi-circle-like is created. The simulation was over 800 steps to show the relative large oval 3.8.

3.4.3 Counter wheel steering

Counter wheel steering is used with the mobile robot SUMMIT. It makes a narrower curve passage possible. This facilitates turning and maneuvering in small areas. The steering angle of front and back wheels are the same.

In figure 3.9 the difference between the diameters is easy to see. By the shorter distance with counter wheel steering the simulation time can be shortened, which also makes the training of the neural network in forthcoming sections easier.

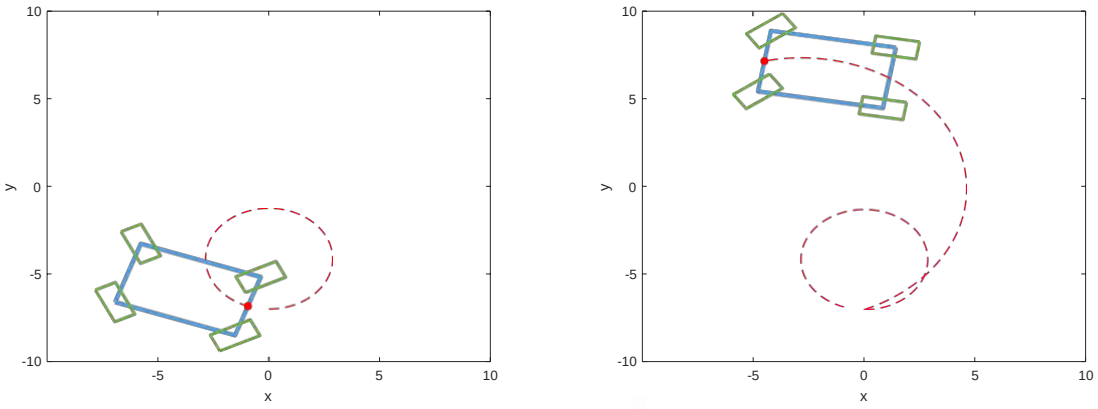


Figure 3.9: Counter wheel steering and front wheel steering

3.4.4 Diagonal steering

The last scenario shows diagonal steering. Front and rear wheels are steered. Because the angles are on opposite sides of the horizontal intersection of the robot, δ_F has to be $-\delta_R$. For this simulation $\delta_F = \frac{\pi}{4}$ and $\delta_R = -\frac{\pi}{4}$. The robot starts on its initial point at $[x, y, \Psi]^T = [-5, -2, 0]^T$. The driven trajectory is shown in 3.10.

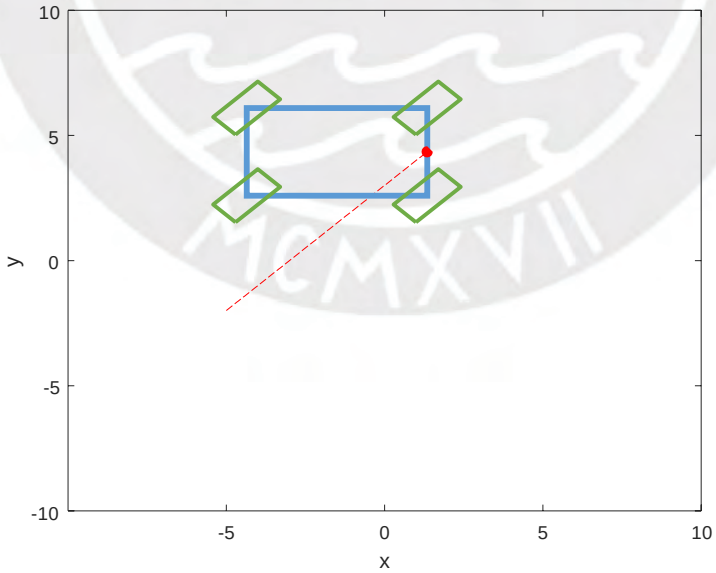


Figure 3.10: Diagonal steering

3.4.5 Conclusions

The general model proposed in section 3.3 gives the opportunity to investigate all kinds of steering, such as front wheel steering, diagonal steering, counter wheel steering, back wheel steering and independent steering (the front axis is independent from the rear axis). The model as presented can perform those different driving styles. The independent steering model is the most complex one, the other models are special cases there of. It will be compared to the more simple model of counter wheel steering.

ET LUX IN TENEBRIS LUCET
MCMXVII

4 Theory of deep learning neural networks

This chapter comprises a brief overview of NN including the structure of a neural network, important functions and different learning strategies. Furthermore the components of the update equations for the training process are developed.

4.1 General overview over neural networks

Neural networks are designed with the idea to imitate the neuron structure of the human brain. While the complete reproduction of a human brain, with its 100 billion neurons, is impossible given the current computational power available, the structure of how information can be processed and analyzed in recreated.

4.1.1 Structure of a neural network

A standard feedforward NN consists of three types of layers— the input layer, the hidden layers and the output layer. The hidden layers can consist of multiple layers. In the following neurons of the same layer have the same activation function. This function can be linear or nonlinear. Usually both the input and output neurons have a linear activation function illustrated in figure 4.1. The activation function of the hidden layers often is nonlinear with m the neuron input and n the output of the neuron, such as the:
sigmoid function (type 1)

$$n = \frac{1}{1 + e^{-m}}, \tag{4.1a}$$

sigmoid function (type 2)

$$n = \frac{2}{1 + e^{-m}} - 1, \tag{4.1b}$$

rectified linear units (ReLU) function

$$n = max(0, m), \tag{4.1c}$$

arcus tangens function

$$n = \arctan(x), \tag{4.1d}$$

and gaussian function

$$n = e^{-m^2} \tag{4.1e}$$

illustrated in 4.2. These different functions can play an important role for the performance of the NN.

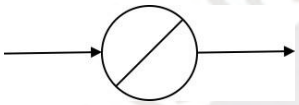


Figure 4.1: Illustration of a linear neuron

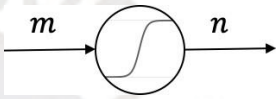


Figure 4.2: Illustration of a nonlinear neuron

The selection for the activation function is determined by any symmetries that may exist within the data, if there is no decision to make try and error will be used.

The neurons are interconnected with a coefficient of connection— the weights. These weights will change during the training process of the NN. It is possible to miss some connections or define weights as constant if needed. In the following, the neural network illustrated in figure 4.3 will be considered.

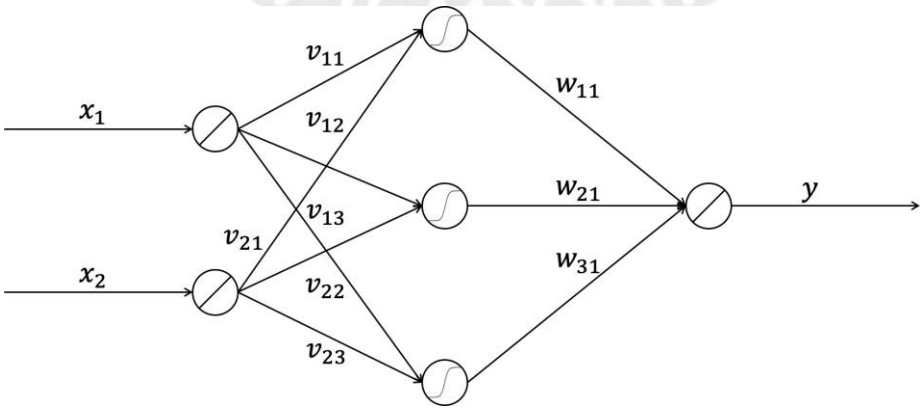


Figure 4.3: Neural Network with one hidden layer

To obtain the output y the inputs have to go through the network beginning with the input layer. The input layer has a linear activation function. To get the input of the hidden layer the input will be multiplied with the weight $v \in \mathbb{R}^{ni \times nh}$, with ni the number of input neurons and nh the number of hidden neurons, at the connection line:

$$\begin{aligned} m_1 &= v_{11}x_1 + v_{21}x_2 \\ m_2 &= v_{12}x_1 + v_{22}x_2 \\ m_3 &= v_{13}x_1 + v_{23}x_2. \end{aligned} \tag{4.2}$$

In figure 4.4 one can see the inputs and outputs of the neurons of the hidden layer.

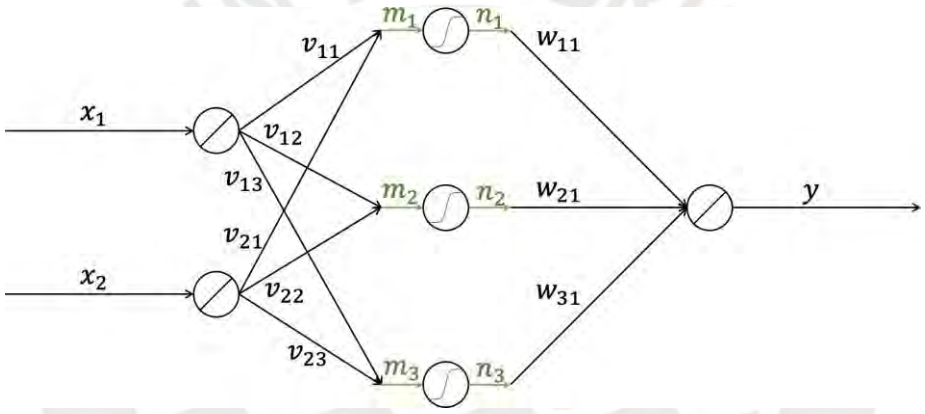


Figure 4.4: Input and Output of the hidden layer

Next an activation function has to be chosen. To generalise the example the activation function will be $f(m)$. So the output of the hidden layer is

$$\begin{aligned} n_1 &= f(m_1) \\ n_2 &= f(m_2) \\ n_3 &= f(m_3). \end{aligned} \tag{4.3}$$

Lastly the output of the neural network will be calculated, by using the weights $w \in \mathbb{R}^{nh \times no}$ with no the number of outputs, as follows:

$$y = w_{11}n_1 + w_{21}n_2 + w_{31}n_3. \tag{4.4}$$

This procedure is called ‘forward pass‘. If the network is well trained the forward pass will be used to calculate the outputs for the process or the control inputs of the model

for the next step.

Before one can use the NN, the first step is to define the structure—requiring the determination of the number of inputs and outputs. Furthermore it has to be decided how many hidden layers with how many neurons are needed to perform the task. Finally, the type of activation function has to be selected.

Lastly, a neuron called bias has to be considered. It is an additional neuron with a constant value. If both inputs are zero but the output is not zero, the given structure of the network with a certain activations function can not produce a value different to zero. Therefore the bias neuron will be added.

4.1.2 Training of a neural network

Other ways in which NN can learn are unsupervised learning, in which systems only receive input data and try to classify it and reinforcement learning, in which a neural network itself can control the input data and receives dynamic output data back together with a task related to this output data (e.g. to maximize a score). In the following supervised learning will be used. So after the neural network structure has been built, each weight receives a random value. This value should be very small. Then the input data is put into the NN and each neuron weights the input signals with its weight and passes the result to the neurons of the next layer. At the output layer, the overall result is then calculated. Usually the first passage has nothing to do with the known actual result, since all neurons have a random initial weight. For supervised training a desired state has to be known. Then the size of the error can be calculated and the part that each neuron had in that error. So the weight of each neuron can be changed in the direction that minimizes the calculated costs with the cost function 4.5. The cost function can be chosen depending on the problem. The easiest way to define the cost function is to sum all the errors for a trajectory or a data set. The cost function used in this thesis, depends on the positioning error and the control variable:

$$J = \frac{1}{2} \sum_{k=0}^{N} \left((x_k - x_k^*)^T Q (x_k - x_k^*) + u_k^T R u_k \right) \tag{4.5}$$

for a data set of N points. Accordingly, the optimization problem describes the weights, which minimize the cost function with:

$$\frac{\partial J}{\partial v_{lij}} = 0, \tag{4.6}$$

where l is the previous layer of the weight and i and j signify the connection between the i-th neuron to the j-th neuron of the next layer. Using the gradient descent method the weighting coefficients will be updated by the following equations

$$v_{lij} = v_{lij} - \eta \frac{\partial J}{\partial v_{lij}}. \tag{4.7a}$$

η is the learning rate of the training and has to be chosen. The update of the weighting coefficients can be made in batches or in every single step.
Training with batches will change the calculation of $\frac{\partial J}{\partial v_{ij}}$ and $\frac{\partial J}{\partial w_{jk}}$. Due to the data will be in batches the average over all the data points will be calculated:

$$\overline{\frac{\partial J}{\partial v_{lij}}} = \frac{\sum_{k=1}^{N} \frac{\partial J}{\partial v_{lij}}}{N}. \tag{4.8a}$$

4.2 Control problem

The objective to be achieved is the autonomous control of positioning of the mobile robot. To control the system properly a neuro controller is designed to provide inputs for the system so that it will respond as required. For this, a NN will be trained to perform as a controller for the mobile robot. The system architecture will be build based on [38]. Figure 4.5 shows the general concept of the control system.

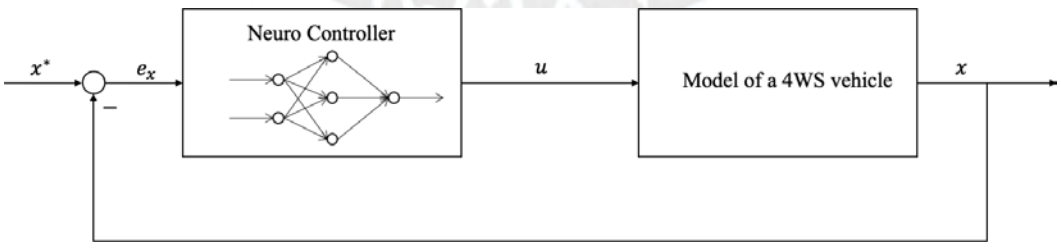


Figure 4.5: Concept of the control system

Like in commonly used control systems, the input of the neuro controller will be the error between desired state x^* and actual state x. The output of the neuro controller

is the input u for the system.
With the nonlinear model of the vehicle given in equations 3.22 a discrete-time state equation

$$x_{k+1} = f(x_k, u_k) \tag{4.9}$$

is used for the control system with $x_k \in \mathbb{R}^3$ and $u \in \mathbb{R}^2$. This results in

$$x_{1(k+1)} = x_{1(k)} + r\frac{\cos(x_{3(k)} + u_1(k)) + \cos(x_{3(k)} - u_2(k))}{2} \tag{4.10a}$$

$$x_{2(k+1)} = x_{2(k)} + r\frac{\sin(x_{3(k)} + u_1(k)) + \sin(x_{3(k)} - u_2(k))}{2} \tag{4.10b}$$

$$x_{3(k+1)} = x_{3(k)} + \frac{r}{L}(\sin(u_1(k)) + \sin(u_2(k))), \tag{4.10c}$$

where k signifies a step. In addition $f(x_k, u_k)$ has to have continuous derivatives. To solve the stabilization problem a control law u_k depending on state x has to be found to accomplish the control objective. The neuro controller represents this law with

$$u_k = k(x_k). \tag{4.11}$$

$k(\cdot)$ is a function $\mathbb{R}^3 \rightarrow \mathbb{R}^2$.

The neuro controller optimizes its cost function of the following quadratic formula:

$$J = \frac{1}{2}\sum_{k=0}^{N}\left((x_k - x_k^*)^T Q(x_k - x_k^*) + u_k^T R u_k\right), \tag{4.12}$$

with Q and R square, symmetric and positive semidefinit matrices with dimensions $Q \in \mathbb{R}^{3\times3}$ and $R \in \mathbb{R}^{2\times2}$. In order to obtain bounded values of the control input u_k, R has to have an inverse.

4.3 Dynamic training of neuro controller

4.3.1 Equations for the training

For the training of the neuro controller, dynamic backpropagation (DBP) will be used. The used training will be gradient descent covered in section 4.1. Therefore the neuro controller will be trained to minimize the cost function given in 4.12 and the weighting

coefficients will be updated using:

$$v = v - \eta \frac{\overline{\partial J}}{\partial v}. \tag{4.13}$$

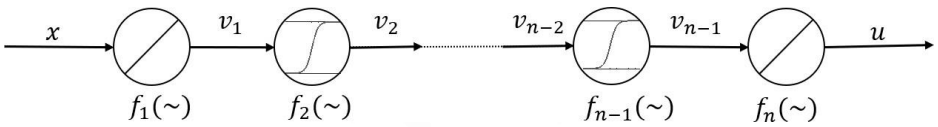


Figure 4.6: Structure of the NN

Where η is the learning rate and $\frac{\overline{\partial J}}{\partial v}$ the total partial derivative of J with respect to weight v. Depending on the number of hidden layers there will be a lot of weighting coefficients updated by a similar function.

In figure 4.6 a general structure of a network is displayed. The model deduced in chapter 3 has three inputs: the position x and y and the angle Ψ. The velocity of the robot will be constant to simplify the network. Thus, there are two outputs δ_F and δ_R. The number of hidden layers as well as the number of neurons in each layer should be changeable, therefore a lot of derivatives have to be considered individually. That means that in figure 4.6 the v_i with $i = 1, \ldots, n$ are matrices with different sizes. To describe a weight between the G-th and the $(G + 1)$-th layer, which connects the i-th neuron with the j-th neuron, there are three indices v_{Gij}.
To describe the total partial derivative of J with respect to v_{Gij} follows:

$$\begin{aligned}\frac{\overline{\partial J}}{\partial v_{Gij}} &= \frac{\partial}{\partial v_{Gij}} \frac{1}{2}\left(\sum_{k=0}^{N} (x_k - x_k^*)^T Q (x_k - x_k^*) + u_k^T R u_k\right) \\ &= \frac{1}{2}\sum_{k=0}^{N} \left(x_k - x_k^*\right)^T Q \frac{\partial \overline{x_k}}{\partial v_{Gij}} + 2 \cdot \frac{1}{2} u^T R \frac{\partial u_k}{\partial v_{Gij}} \\ &= \sum_{k=0}^{N} \left((x_k - x_k^*)^T Q \frac{\partial \overline{x_k}}{\partial v_{Gij}} + u_k^T R \frac{\partial \overline{u_k}}{\partial v_{Gij}}\right).\end{aligned} \tag{4.14}$$

The derivatives included in equation 4.14 can be divided into model-depending and network-depending.

4.3.2 Terms of the derivative of the cost function

$\overline{\frac{\partial x_k}{\partial v_{Gij}}}$ is the total derivative of x_k with respect to the weighting coefficient. To update the value a recursive equation will be used:

$$\overline{\frac{\partial x_{k+1}}{\partial v_{Gij}}} = \frac{\partial x_{k+1}}{\partial u_k}\frac{\partial u_k}{\partial v_{Gij}} + \left(\frac{\partial x_{k+1}}{\partial x_k} + \frac{\partial x_{k+1}}{\partial u_k}\frac{\partial u_k}{\partial x_k}\right)\overline{\frac{\partial x_k}{\partial v_{Gij}}}. \tag{4.15}$$

Similar the derivative of $\overline{\frac{\partial u_k}{\partial v_{Gij}}}$ is calculated with the recursive equation:

$$\overline{\frac{\partial u_{k+1}}{\partial v_{Gij}}} = \frac{\partial u_{k+1}}{\partial v_{Gij}} + \frac{\partial u_{k+1}}{\partial x_k}\overline{\frac{\partial x_k}{\partial v_{Gij}}}. \tag{4.16}$$

$\frac{\partial x_{k+1}}{\partial u_k}$ and $\frac{\partial x_{k+1}}{\partial x_k}$ are derivatives, which depend on the model equations. The model equations are deduced in chapter 3.3. With the equations 4.10, $\frac{\partial x_{k+1}}{\partial u_k}$ can be computed as:

$$\begin{aligned}\frac{\partial x_{k+1}}{\partial u_k} &= \begin{bmatrix} \frac{\partial x_{1(k+1)}}{\partial u_{1(k)}} & \frac{\partial x_{1(k+1)}}{\partial u_{2(k)}} \\ \frac{\partial x_{2(k+1)}}{\partial u_{1(k)}} & \frac{\partial x_{2(k+1)}}{\partial u_{2(k)}} \\ \frac{\partial x_{3(k+1)}}{\partial u_{1(k)}} & \frac{\partial x_{3(k+1)}}{\partial u_{2(k)}} \end{bmatrix} \\ &= \begin{bmatrix} -\frac{1}{2}r\sin(u_1 + x_3) & -\frac{1}{2}v\sin(u_2 - x_3) \\ \frac{1}{2}r\cos(u_1 + x_3) & -\frac{1}{2}v\cos(u_2 - x_3) \\ \frac{1}{2}\cos(u_1) & \frac{1}{2}\cos(u_2) \end{bmatrix}\end{aligned} \tag{4.17}$$

with $u_k = (u_{1(k)}, u_{2(k)})^T = (\delta_F, \delta_R)^T$ and $x_k = (x_{1(k)}, x_{2(k)}, x_{3(k)})^T = (x, y, \Psi)^T$. Thus with the equations 4.10 $\frac{\partial x_{k+1}}{\partial x_k}$ can be computed as:

$$\begin{aligned}\frac{\partial x_{k+1}}{\partial x_k} &= \begin{bmatrix} \frac{\partial x_{1(k+1)}}{\partial x_{1(k)}} & \frac{\partial x_{1(k+1)}}{\partial x_{2(k)}} & \frac{\partial x_{1(k+1)}}{\partial x_{3(k)}} \\ \frac{\partial x_{2(k+1)}}{\partial x_{1(k)}} & \frac{\partial x_{2(k+1)}}{\partial x_{2(k)}} & \frac{\partial x_{2(k+1)}}{\partial x_{3(k)}} \\ \frac{\partial x_{3(k+1)}}{\partial x_{1(k)}} & \frac{\partial x_{3(k+1)}}{\partial x_{2(k)}} & \frac{\partial x_{3(k+1)}}{\partial x_{3(k)}} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & \frac{1}{2}r(\sin(u_2 - x_3) - \sin(u_1 + x_3)) \\ 0 & 1 & \frac{1}{2}r(\cos(u_2 - x_3) + \cos(u_1 + x_3)) \\ 0 & 0 & 1 \end{bmatrix}.\end{aligned} \tag{4.18}$$

To describe the equations 4.16 and 4.15 fully, two other terms have to be considered. They depend on the structure of the network. In the following sections, the general equations for the calculation of these derivatives are explained at each point in the neural network, i.e. according to any weight.

State derivative of the output $\frac{\partial u_r}{\partial x_m}$

In order to ensure the generality of the neural network, the derivatives must be derived individually. The indices will be changed for the following considerations to the component indices in step k. Therefore $\frac{\partial u_r}{\partial x_m}$ is to be describe for a network with s hidden layers. The approach is to consider the problem with a few hidden layers and then generalize it.
The notation of the problem is as follows:

$$\frac{\partial u_r}{\partial x_m} \overset{\cdot}{\underset{0}{\cdot}} = H(r,m)_{\cdot_0} = v_{1mr}. \tag{4.19}$$

This means that the derivative of $\frac{\partial u_r}{\partial x_m}$ with zero hidden layers can be described through the weight of the first space between input and output, which connects the m-th input with the r-th output. The layers are connected through weights with three indices. The first one describes the previous layer. Signified by the layer in which the base of the arrow begins in figure 4.7. The second and third indices describe which neurons the weight connects. So the second index is the number of the neuron where the arrow starts and the third, the number of the neuron where it ends in the next layer. The number of neurons in the n-th layer is described through $\#N(n)$. All neurons in one layer have the same activation function, which is described by $f_l(\sim)$ with the layer l. The derivation of the activation function in the l-th layer on the o-th neuron is described by $f'_l(o)$. The function $H(r,m)$ describes the reduced notation.
$_0$

The equation 4.19 results from the equation of u. In figure 4.7 one can see a general neural network with three inputs and two outputs. There is no hidden layer accordingly for example u_1 can be described as:

$$u_1 = x_1 v_{111} + x_2 v_{121} + x_3 v_{131}. \tag{4.20}$$

So that

$$\frac{\partial u_1}{\partial x_2} = v_{121}. \tag{4.21}$$

In general this means

$$\frac{\partial u_r}{\partial x_m} = v_{1mr}. \tag{4.22}$$

Now a network with one hidden layer is considered.The network in figure 4.8 is used as an example. This network has one hidden layer with 4 neurons.

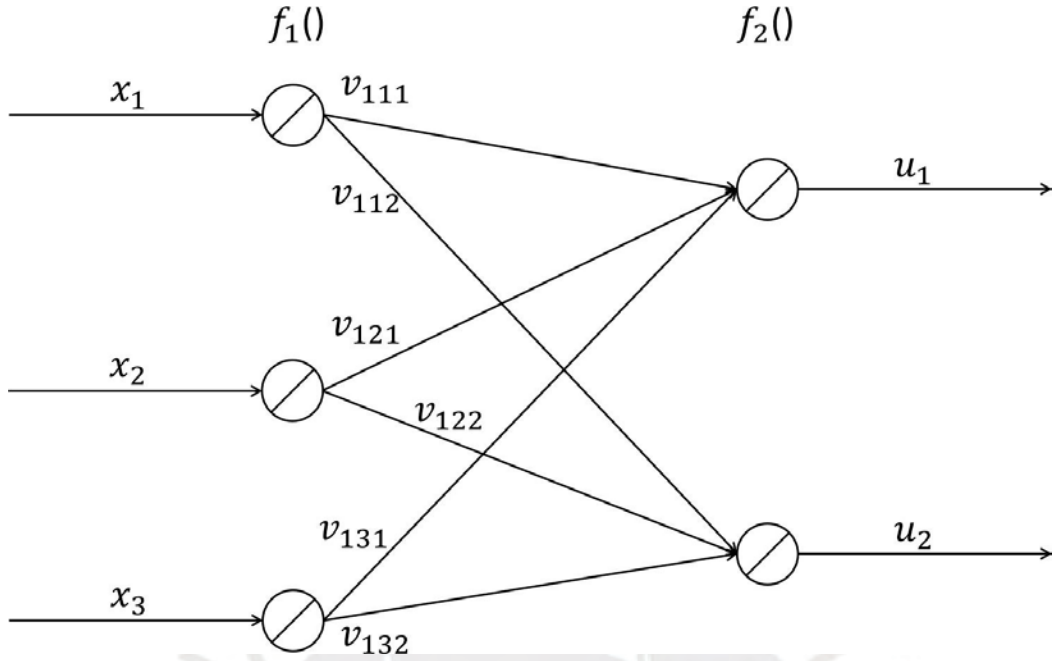


Figure 4.7: Structure of the NN with no hidden layer

Also, in this case, it is possible to set up the equations for u. In this case, one equation will suffice:

$$\begin{aligned} u_1 = & v_{211} f_2(x_1 v_{111} + x_2 v_{121} + x_3 v_{131}) + v_{221} f_2(x_1 v_{112} + x_2 v_{122} + x_3 v_{132}) + \\ & v_{231} f_2(x_1 v_{113} + x_2 v_{123} + x_3 v_{133}) + v_{241} f_2(x_1 v_{114} + x_2 v_{124} + x_3 v_{134}). \end{aligned} \tag{4.23}$$

Again, it is possible to derive the equation

$$\begin{aligned} \frac{\partial u_1}{\partial x_2} &= v_{211} f_2^j(1) v_{111} + v_{221} f_2^j(2) v_{112} + v_{231} f_2^j(3) v_{113} + v_{241} f_2^j(4) v_{114} \\ &= \sum_{i=1}^{4} f_2^j(i) v_{11i} v_{2i1}. \end{aligned} \tag{4.24}$$

This leads to:

$$\frac{\partial u_r}{\partial x_m} = H(r,m) = \sum_{j=1}^{\#N(1)} f_2^j(j) v_{2jr} v_{1mj}. \tag{4.25}$$

In the case of two hidden layers the representation of the NN and the equations become very complex. In this example, shown in figure 4.9, the network has two hidden layers. The first hidden layer has 4 neurons and the second one 3 neurons. So for u_1 the result

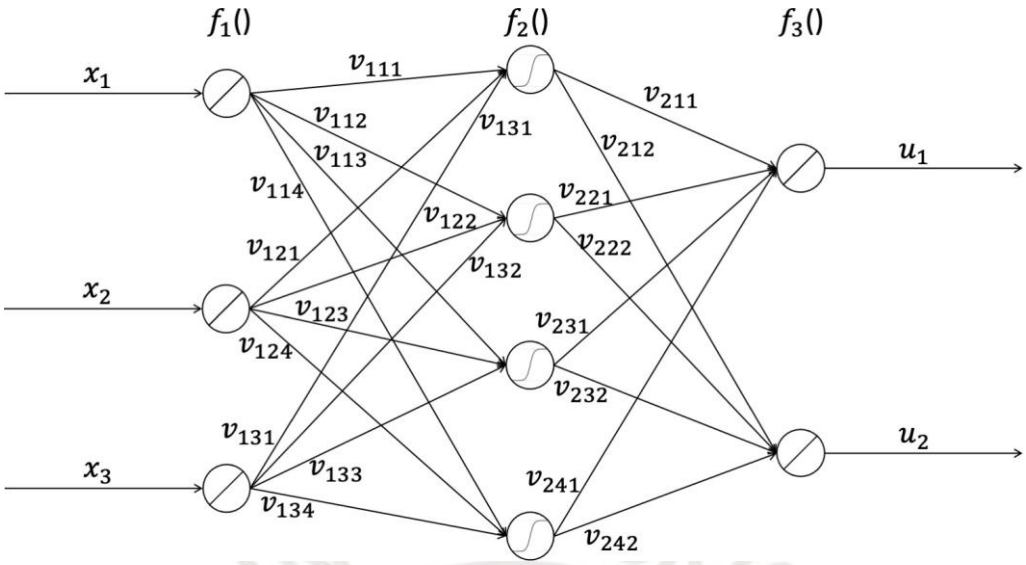


Figure 4.8: Structure of the NN with one hidden layer

is:

$$\begin{aligned} u_1 = & f_3(f_2(x_1v_{111} + x_2v_{121} + x_3v_{131})v_{211} + f_2(x_1v_{112} + x_2v_{122} + x_3v_{132})v_{221} + \\ & f_2(x_1v_{113} + x_2v_{123} + x_3v_{133})v_{231} + f_2(x_1v_{114} + x_2v_{124} + x_3v_{134})v_{241})v_{311} + \\ & f_3(f_2(x_1v_{111} + x_2v_{121} + x_3v_{131})v_{212} + f_2(x_1v_{112} + x_2v_{122} + x_3v_{132})v_{222} + \\ & f_2(x_1v_{113} + x_2v_{123} + x_3v_{133})v_{232} + f_2(x_1v_{114} + x_2v_{124} + x_3v_{134})v_{242})v_{321} + \\ & f_3(f_2(x_1v_{111} + x_2v_{121} + x_3v_{131})v_{213} + f_2(x_1v_{112} + x_2v_{122} + x_3v_{132})v_{223} + \\ & f_2(x_1v_{113} + x_2v_{123} + x_3v_{133})v_{233} + f_2(x_1v_{114} + x_2v_{124} + x_3v_{134})v_{243})v_{331}. \end{aligned} \tag{4.26}$$

Since nearly every term depends on x the derivative is

$$\frac{\partial u_1}{\partial x_2} = \sum_{i=1}^{3} [f_3^{j}(i)v_{3i1}(\sum_{j=1}^{4} f_2^{j}(j)v_{2ji} \quad v_{12j})]. \tag{4.27}$$

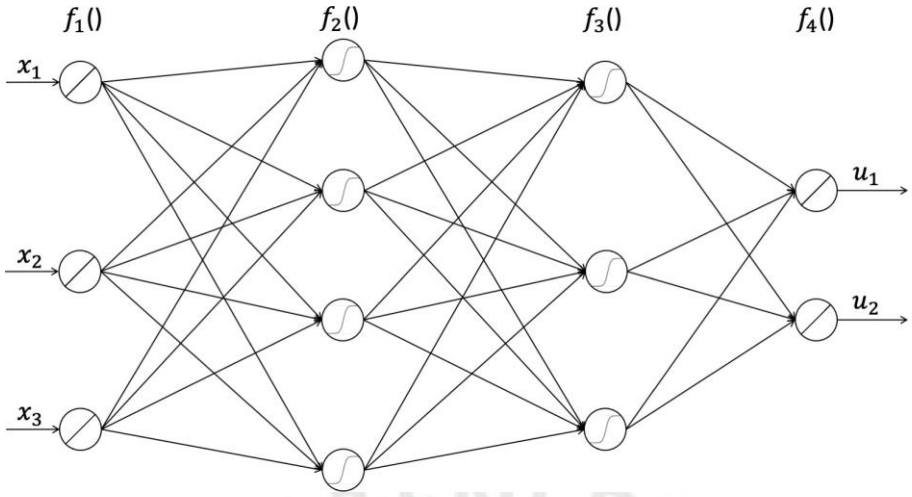


Figure 4.9: Structure of the NN with two hidden layers

In general for two hidden layers follows:

$$\frac{\partial u_r}{\partial x_m} = H(r,m)_2 = \sum_{i=1}^{\#N(2)} [f_3(i)\, v_{3ir} \underbrace{\left(\sum_{j=1}^{\#N(1)} f_2^j(j)\, v_{2ji} v_{1mj} \right)}_{H(i,m)_1}]. \tag{4.28}$$

Using the knowledge obtained from equations 4.27 and 4.28, the following equation can be set up for an undefined number s of hidden layers:

$$\frac{\partial u_r}{\partial x_m} = H(r,m)_s = \sum_{k=1}^{\#N(s)} [f_{s+1}^j(k) v_{(s+1)kr} H(k,m)_{s-1}]. \tag{4.29}$$

Weight derivative of the output $\frac{\partial u_i}{\partial v_{Gkj}}$

To describe the term $\frac{\partial u_i}{\partial v_{Gkj}}$ in general for an unknown number of hidden layers an example is given to illustrate the problem in a smaller environment and then generalized. Also, here the example will be the network shown in figure 4.9.

The goal of these considerations is to be able to show the derivations of the outputs to all weights in all layers. For this purpose work is done from the back to the front of the NN, because the longer the path from input to output, the more complex the equation becomes. At this point the previous notation is extended by another variable out_{Gj} which is the output of the j-th neuron in the G-th layer.

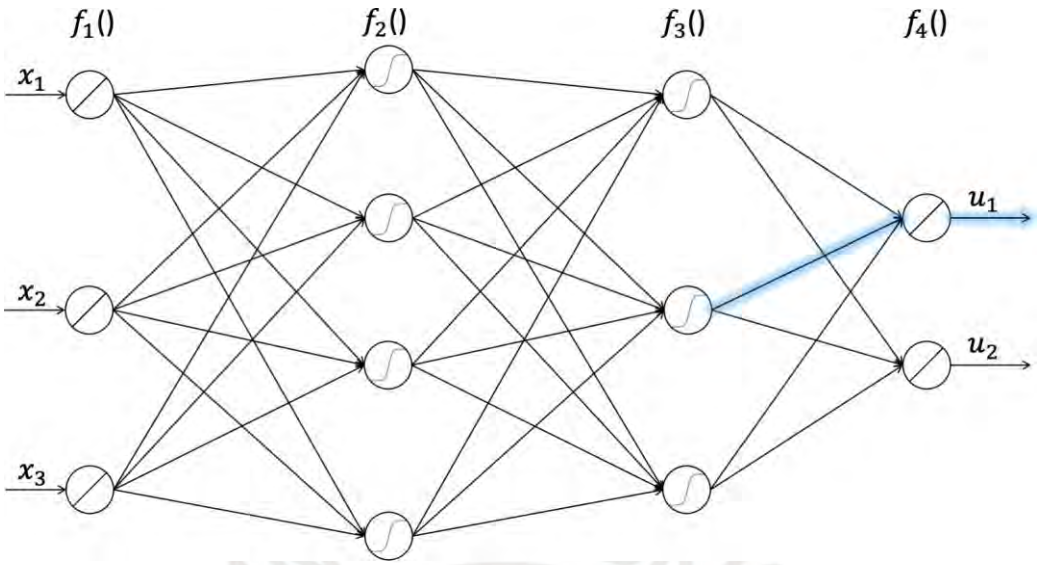


Figure 4.10: Structure of the NN highlighting $\frac{\partial u1}{\partial v_{321}}$

Kronecker delta

The Kronecker delta is a mathematical symbol and is defined as:

$$\delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}. \tag{4.30}$$

In the last hidden layer, $\frac{\partial u_i}{\partial v_{3kj}}$ has to be calculated. As demonstrated in figure 4.10 there are three possible neurons which can lead to a specific output. Only the weights, which connect to the right u_i will count. This means that

$$\frac{\partial u_i}{\partial v_{3kj}} = out_{3k}\,\delta_{ij}, \tag{4.31}$$

with δ_{ij} the Kronecker delta. If the weight does not connect to the right output u_i the derivative will be zero.

In the penultimate hidden layer only one way through the network will lead to a derivative different from zero. It is shown in figure 4.11, which illustrates $\frac{\partial u_1}{\partial v_{222}}$.

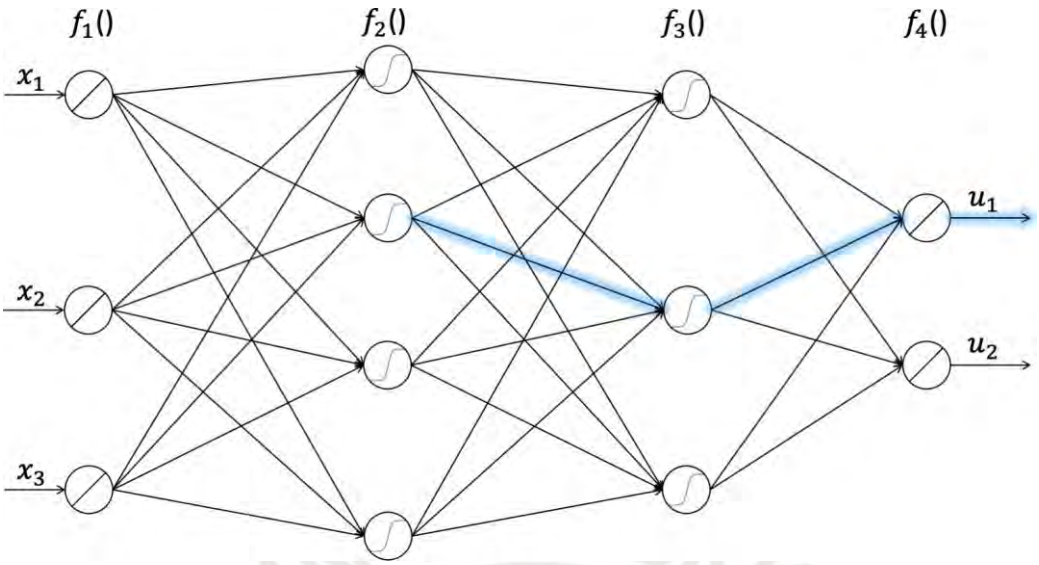


Figure 4.11: Structure of the NN highlighting $\frac{\partial u_1}{\partial v_{222}}$

This means starting with the output of the neuron, where the weight starts out_{22} the derivation depends on f_3^j (2) and the connecting weight to u_1, which is v_{321}. out_12 describes the output of the second neuron of the first layer. Therefore applies the equation:

$$\frac{\partial u_1}{\partial v_{222}} = out_{22} f_3^j(2) v_{321}. \tag{4.32}$$

For the upcoming generalization applies

$$\frac{\partial u_i}{\partial v_{2kj}} = out_{2k} \underbrace{f_3^j(j) v_{3ji}}_{=:M_3(j,i)}. \tag{4.33}$$

Therefore $M_3(j, i)$ is a variable, which describes the last two factors. It is this definition that allows for the generalization of the equation later. One layer further back, as demonstrated in figure 4.12, all paths are weightmarked, which will influence the derivation of $\frac{\partial u_1}{\partial v_{112}}$. The connecting weight between the first and the second layer is fixed, but after that there are three ways to reach u_1. All of those paths have to be investigated. Considering just the blue path in the middle of the network. The derivation will depend on out_{11}, f_2^j (2), v_{222}, f_3^j (2) and v_{321}. The upper green path will depend on out_{11}, f_2^j (2), v_{221}, f_3^j (1) and v_{311}. As well as the other green part will depend on out_{11}, $f_2^j(2)$, v_{223}, $f_3^j(3)$ and v_{331}. Thus the equation is

$$\frac{\partial u_1}{\partial v_{112}} = out_{11} f_2^j(2) \sum_{p=1}^{3} f_3^j(p) v_{3\,1p} v_{22\,p} \tag{4.34}$$

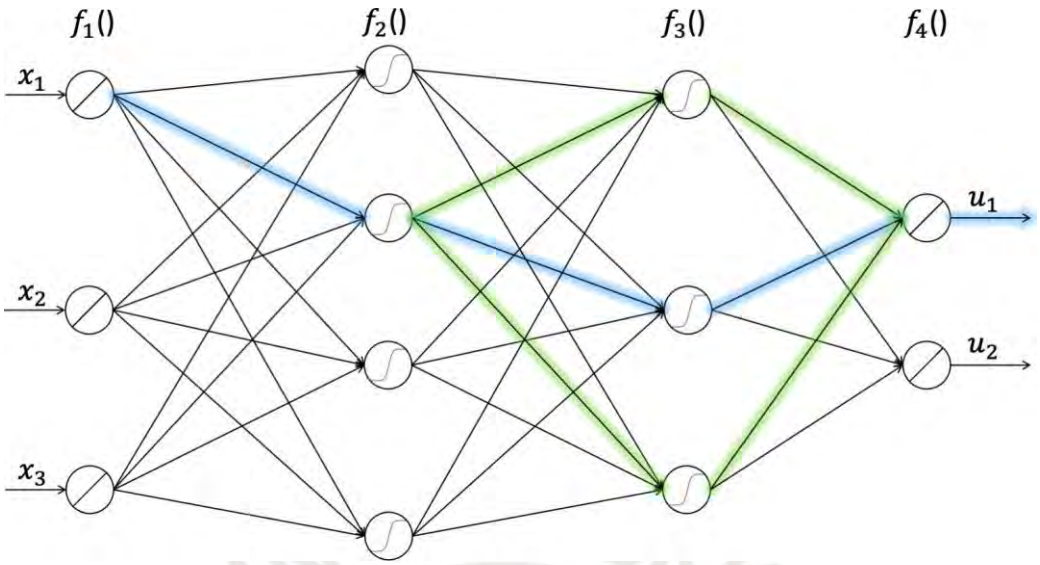


Figure 4.12: Structure of the NN highlighting $\frac{\partial u_1}{\partial v_{122}}$ and all possible ways through the network

The generalization is

$$\frac{\partial u_i}{\partial v_{1kj}} = out_{1k} f_2^j(j) \underbrace{\sum_{p=1}^{\#N(3)} f_3^j(p) v_{3p1} v_{22p}}_{=:M_2(j,i)} . \tag{4.35}$$

To understand the generalisation step better in the following the equation for one more hidden layer is written down:

$$\frac{\partial u_i}{\partial v_{0kj}} = out_{0k} f_1^j(j) \underbrace{\sum_{m=1}^{\#N(2)} f_2^j(m) \underbrace{\sum_{p=1}^{\#N(3)} f_3^j(p) v_{3pi} v_{2mp}}_{M_3(p,i)}}_{M_2(m,i)} v_{1jm} \underbrace{\cdot}_{=:M_1(j,i)} . \tag{4.36}$$

Thus an equation can be set up, which is valid for all hidden layers except for the last

$$\frac{\partial u_i}{\partial v_{Gkj}} = out_{Gk} f_{(G+1)}^j(j) \sum_{p=1}^{\#N(G+2)} M_{(G+2)}(p,i) v_{(G+1)jp} . \tag{4.37}$$

This can be explained by the additional paths if one hidden layer further towards the entrance is considered. With every step one sum is added for all the neurons, which possibly could bring the result to the desired output.

The last hidden layer in the network holds another equation:

$$\frac{\partial u_i}{\partial v_{Gkj}} = out_{Gk}. \tag{4.38}$$

But only if G = numberOfLayers − 1. The recursive equation for M finally is

$$M_{(G+1)}(j,i) = f'_{(G+1)}(j) \sum_{p=1}^{\#N(G+2)} M_{(G+2)}(p,i) v_{(G+1)jp}. \tag{4.39}$$

4.3.3 The challenge of overfitting or underfitting

During the training of neural networks, training challenges can arise in addition to the parameterization of the network. Overfitting and underfitting can be consequences of the training process. While underfitting may indicate that the training was too short so that important influencing factors were not taken into account, overfitting means that the network only memorizes the training samples so that a transfer to the generality is not possible.
The main causes of underfitting are the unfitting of learning algorithm and model complexity as well as the under-consideration of influencing factors. An example can be a constant value as an output although the input changes. If underfitting occurs, the learning algorithm and its parameterization should be reconsidered.

Overfitting describes the opposite effect. In this case, the model is specialized to the training data. This achieves a high model quality in the first sense, but since the model is overfitted, a transfer of the model to the generality is no longer possible. In figure 4.13 one can see an example of overfitting. While in case B the cost function decreases, another training step with case A means, that the cost will increase. The goal is to find the point, where case A also has its minimum.
The challenge of training is to avoid overfitting and underfitting. Experience, background knowledge and expertise help to find a good applicable model. To avoid overfitting, an earlier stop point in the iterative training of this network should be considered.

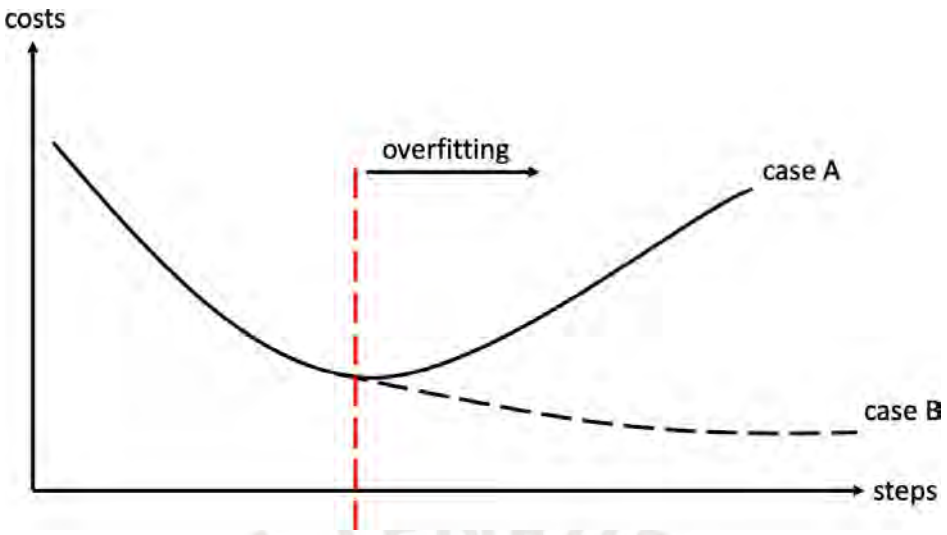


Figure 4.13: Example for overfitting

For the algorithm of the thesis two arrangements were found. The network will be saved if the costs are less than those in the previous step. To make further incremental learning possible despite higher error, the minimum cost is reset to a high value in each increment step. This reduces the possibility of overfitting in order to continue a reduction in error.

4.4 Incremental learning

To solve the control problem supervised learning including DBP is used. But after choosing a learning method also a paradigm about the tasks can be chosen. This process is shown in figure 4.14.

4.4.1 Incremental learning tasks

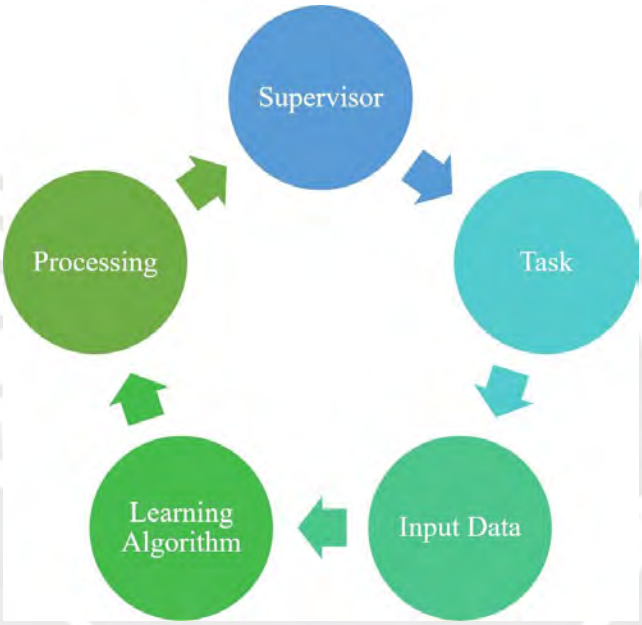


Figure 4.14: Process of Incremental Learning

Incremental Learning (IL) is a dynamic method in machine learning. The idea is derived from human learning, as children also solve simple tasks at first and master more difficult ones over time. The learning process takes place when a modification to the previous learned example is made. The most important difference to traditional machine learning is that the training data appear over time and are not available at the beginning [39]. Thereby it is possible to change the task or complicate the task.

In the context of the control problem this means, that the robot should learn first from an easy mission. Therefore the initial point is very close to the desired point. After learning the most simple task, the initial point will be more advanced. The angle Ψ can be changed or the distance to the desired position. Possible steps are illustrated

in figure 4.15, where the desired position is $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_{pos} \\ y_{pos} \\ \Psi \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$.

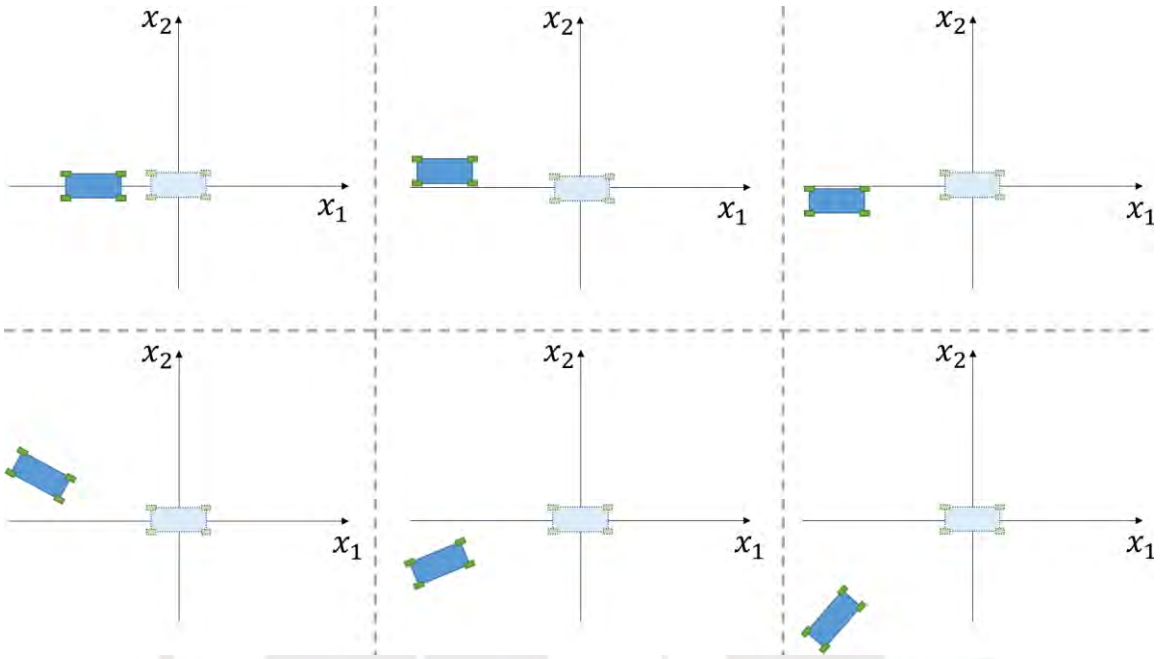


Figure 4.15: Incremental Learning Tasks

Importantly, changes in the initial points are only useful when the algorithm is converging. If the algorithm does not converge, the initial point must be closer to the target.

During the implementation, the initial points were defined and trained. The number of points and orientations will be used in an algorithm which calculates the initial states. A distinction must also be made between the two model types. The model with independent steering will solve the tasks differently than the model with 4WS.

4.4.2 Initial points

To train the network properly a selection of initial points have to be found. The goal is to achieve good results with few initial points. There are many approaches to find those sets. In the following one of those approaches, which will be used in this thesis, will be described.

The main idea is to start at two mirror points with different orientations. To simplify the data the initial position in *y*-direction is always zero. Also the initial orientation of the robot will be $\Psi = \frac{\pi}{2}$. In a set of distances the *x*-coordinates will be described. The values increase with the index.

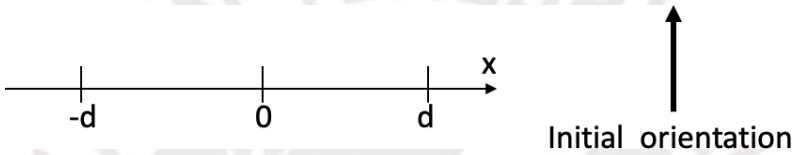


Figure 4.16: Initial Points

In figure 4.17 the different orientations are illustrated. Using an algorithm, the initial states can be modified due to varying numbers of different orientations.

for $i = 1 : numberOrientationPoints$

$\Psi_{initial} = \frac{\pi}{2} + \frac{2\pi(i-1)}{numberOrientationPoints}$

$x_{right}(1, i) = d$

$x_{right}(2, i) = \Psi_{initial}$

$x_{left}(1, i) = -d$

$x_{left}(2, i) = \Psi_{initial}$

end

With this algorithm, the initial points with a given distance to zero will be calculated. Note that $numberOrientationPoints$ is the number of half of the initial points. It describes the number of orientations of each site. Also it means that the full circle will be parted in $numberOrientationPoints$ to obtain the intial orientations similar to figure 4.17.

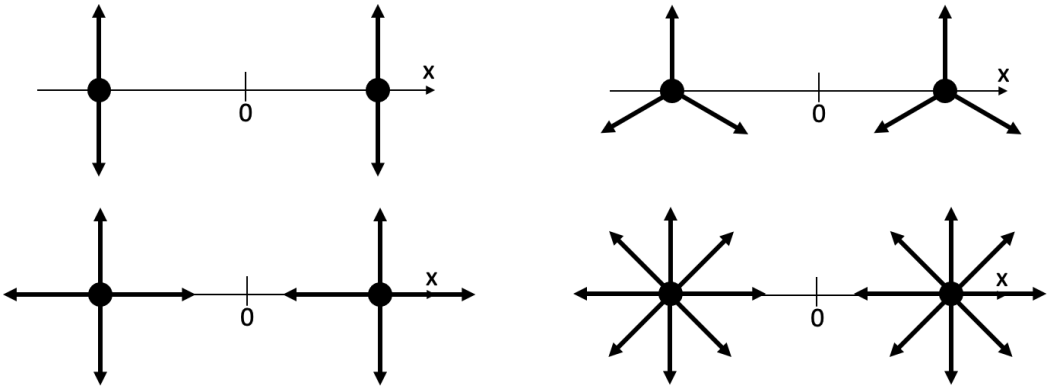


Figure 4.17: Initial Orientations

Ultimately, the incremental learning points take place at different distances to zero and with a given number of orientations.

ET LUX IN TENEBRIS LUCET
MCMXVII

5 Implementation of the system

For the implementation of the system an object-oriented approach was chosen. The implementation is realized with the software MATLAB®. In order for different training scenarios to be represented, a more general approach is used.

5.1 Architecture

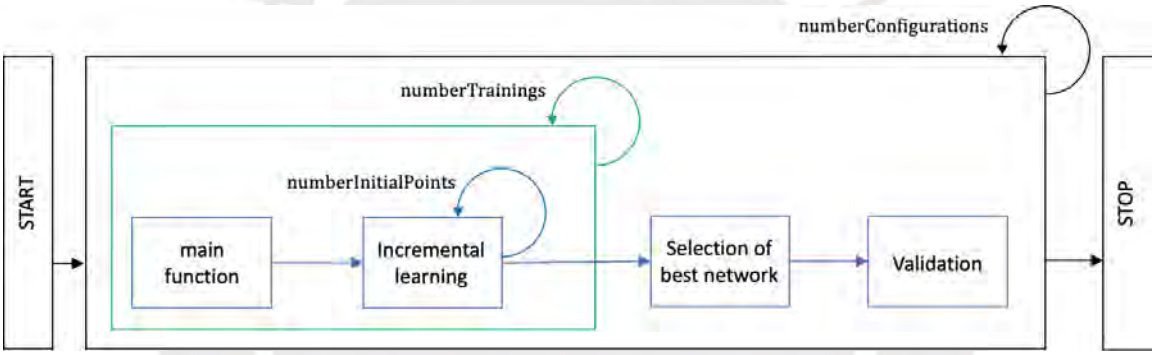


Figure 5.1: Block diagram of neural network training process

The system consists of four different parts, which can be seen in figure 5.1. The main function, alongside a configuration set, parameterizes the network, model and incremental strategy. Then the incremental learning function will be called, which trains the network for the different initial points. After the network is trained several times, an algorithm is called to find the best trained network for this particular configuration set. This network will be used for the validation process.

In the following sections the functions, classes and parameters as well as the algorithms of each system part will be examined more closely. Therefore an overview is given in figure 5.2.

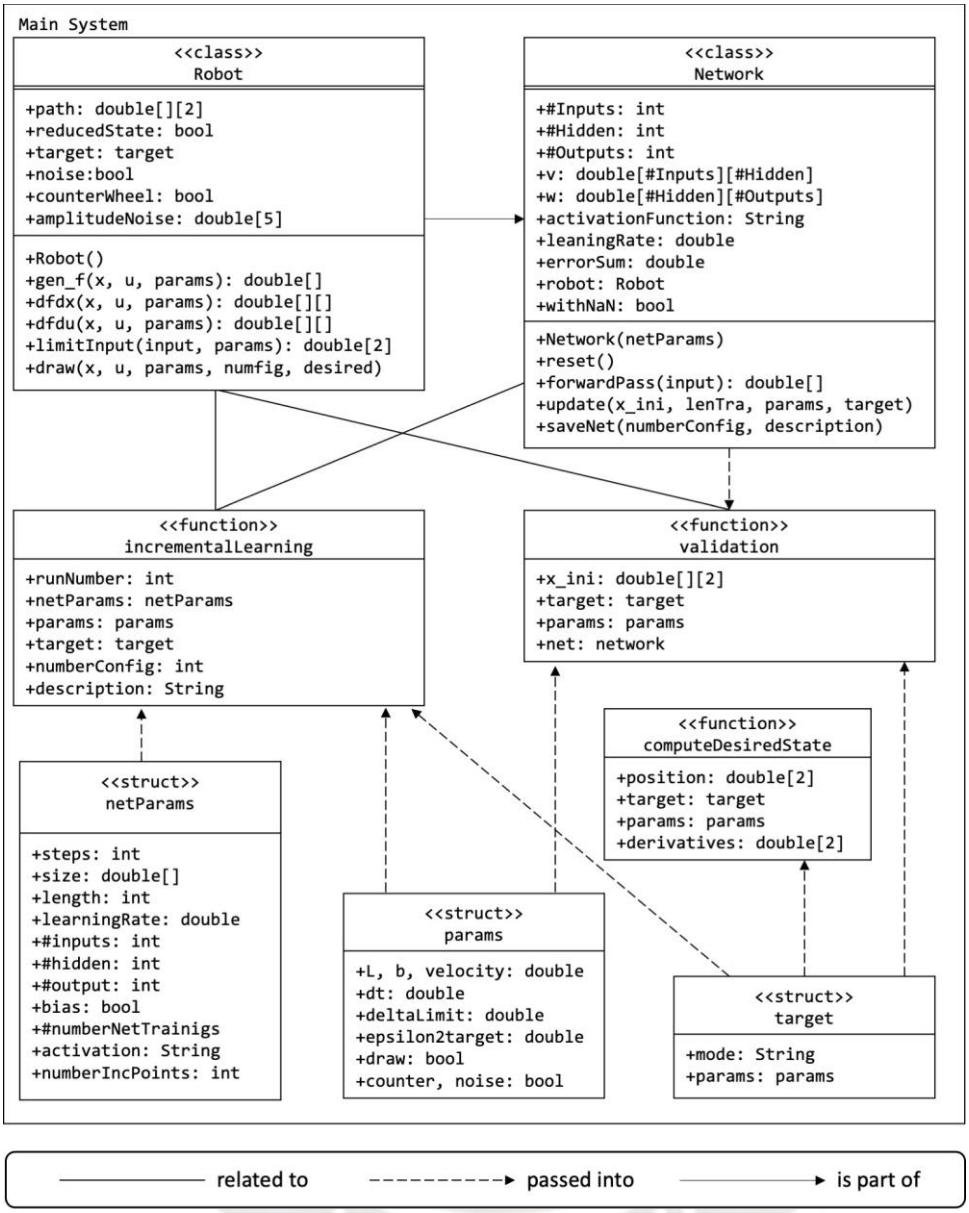


Figure 5.2: Functions, classes and parameters of the system

5.2 Structures in use for the system

In figure 5.2 there are the different *structs* which store the parameters for the system. While *params* hold the model properties, *netParams* and *target* are important for the training and validation. *netParams* encapsulates all parameters which describe the structure of the network and the parameters for the incremental learning. In *target*, the parameters for the target trajectory will be stored.

Implementation of the model

Part of the system is the model of a 4WS vehicle, designed in chapter 3. Therefore a class *Robot* is introduced. Because the visualisation is an object method of *Robot*, properties like the driven path and the target are saved. Next to the constructor there are methods to calculate the derivatives.

5.2.1 Model functions

The object functions of the class *Robot* are functions to calculate the derivatives as $\frac{\partial x_{k+1}}{\partial u_k}$, $\frac{\partial x_{k+1}}{\partial x_k}$ and $\dot{x}$, but also functions to visualize the robot itself. The equations can be found in 4.17, 4.18 and 3.22.

5.2.2 Parts of the model

Counter wheel steering

As described in chapter 3 a special case of 4WS is counter wheel steering. Here the angles δ_F and δ_R are equal. In the class *Robot* there is a property named *counterwheel*, which is a boolean. If it is true the model will change to the model in equation 3.25.

Reduced state

Like mentioned in chapter 3.3 there are useful cases to run the system with a reduced state to simplify the calculations. Therefore the variable reducedState as a boolean can change the model as well. While the normal state consists of three parts: $\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix} = \begin{pmatrix} x \\ \Psi \\ \dot{\Psi} \end{pmatrix}$, the reduced state will be $\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} x \\ \Psi \end{pmatrix}$.

5.2.3 Visualization of the model

The visualization of the robot takes place in the function draw(). This function is part of the object methods of the class *Robot*. The process, also shown in figure 5.3, holds three steps:

1. drawing the robot as a rectangle at point (0,0),
2. rotating the points by angle Ψ,

3. shifting the rectangle to the desired position.

In the first step five x-coordinates and five y-coordinates have to be found to describe the rectangle. The rectangle of the robot is located in (0,0) with its center. Therefor the coordinates depend of the length and width of the robot. It follows that $rect_x = [-\frac{L}{2}, \frac{L}{2}, \frac{L}{2}, -\frac{L}{2}, -\frac{L}{2}]$ and $rect_y = [-\frac{b}{2}, -\frac{b}{2}, \frac{b}{2}, \frac{b}{2}, -\frac{b}{2}]$. The same idea is used to create a rectangle for the wheels.
The second step includes the rotation of every point y an angle. Therefore a rotation matrix like in [40] is used to rotate the coordinates.

Rotation Matrix

$$\begin{pmatrix} x^j \\ y^j \end{pmatrix} = \begin{pmatrix} \cos(\Phi) & \sin(\Phi) \\ -\sin(\Phi) & \cos(\Phi) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{5.1}$$

Ultimately, a new position is calculated and in the last step shifted to the new position. Shifting means to sum the resulting vectors.

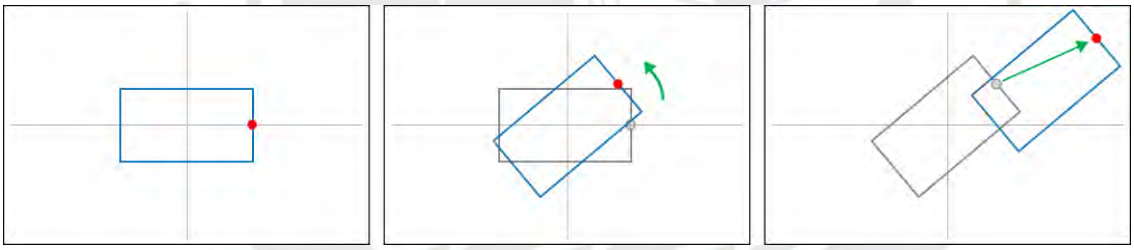


Figure 5.3: Steps for visualization of robot

Part of the visualization is also the functions of the desired trajectories and the desired state itself. For the validation of the networks a draw fast method is used to show the calculated trajectory of the network without the visualization of the robot itself.
For the simulation the body of the vehicle is a blue rectangle with a red dot to show the orientation. Also the driven trajectory is shown with a red line. The wheels are displayed with green rectangles. Part of the evaluation is the usage of an ϵ-area around the desired trajectory. Therefore an orange tube is displayed around the desired trajectory in green, what can be seen in figure 5.4.

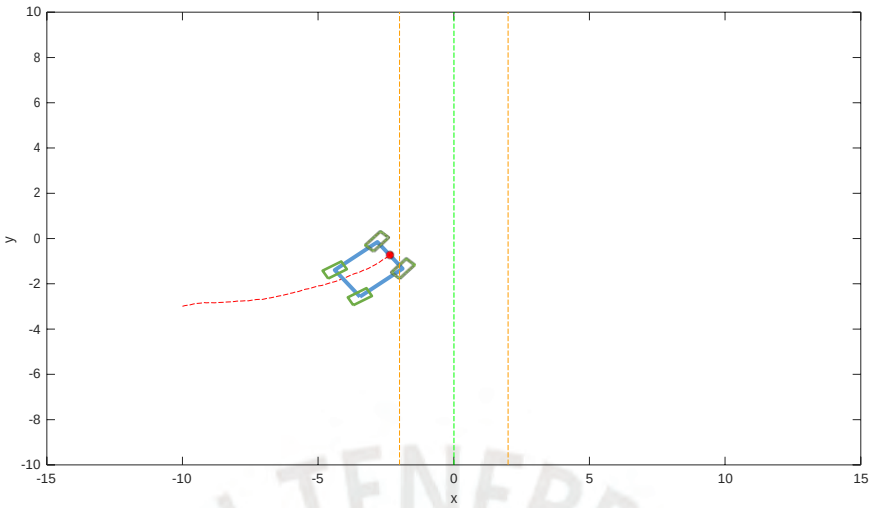


Figure 5.4: Visualization of robot while upwards driving

For the scenarios of straight line following and circle line following the desired state of the full vehicle is drawn in grey, shown in figure 5.5.

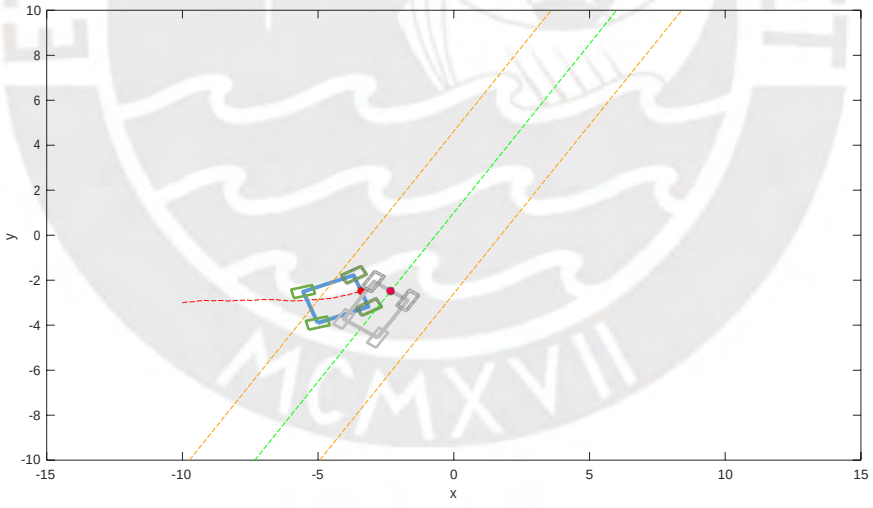


Figure 5.5: Visualization of robot while following a straight line

The advantage is, that a statement about the current situation can be made more easily, when the desired state is visible.

5.3 Implementation of the learning algorithm

The learning algorithm is implemented by the function *incrementalLearning()* and the object methods of the class *Network*. In figure 5.6 the structure of the algorithm is displayed. To describe the algorithm more closely the functions *incrementalLearning()* and *update()*, shown in figure 5.2, have to be examined. Another function, which is used in the object method *update()* is the *forwardPass()* function.

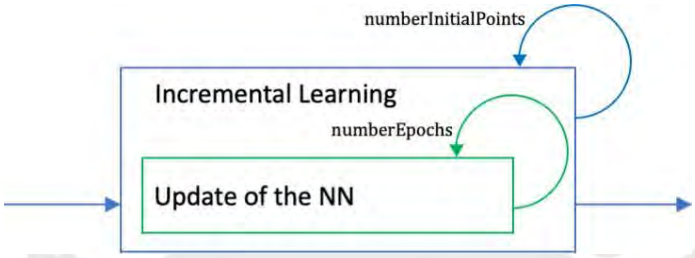


Figure 5.6: Structure of the learning Algorithm

Forward pass

The *forwardPass()* function uses the equation 4.4 to calculate the output of the network with one hidden layer. If the network is generalized, more structures must be given to store the values. In figure 5.7 a general network is given. The layers are presented through one neuron, which stands for all the neurons of the layer. From this, one can see that the various activation functions and weight matrices require their own structures such as tensors to store all the data.

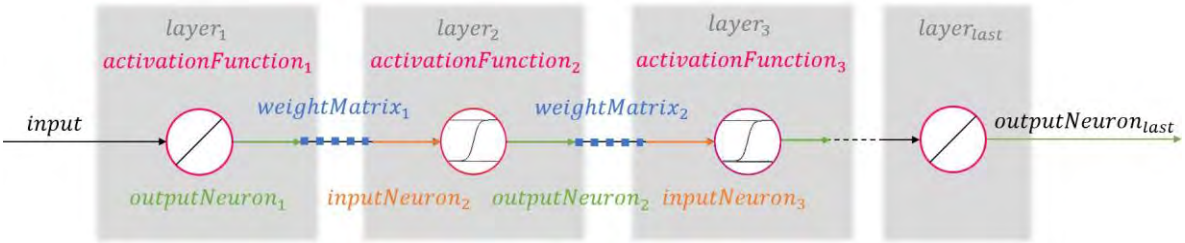


Figure 5.7: NN structure and variables

Incremental learning

The *incrementalLearning()* function consists of two parts: the function call of *update()* and the calculation of all the incremental points for a given distance in x-direction to zero. To calculate the initial states the algorithm introduced in section 4.4.2 will

be used. The *update()* function will be called as often as necessary until all initial points have been used. Also the number of epochs influences the number of calls of the *update()* function. That means, that the network will be trained for a certain number of trajectory steps and overall in a certain number of epochs, to reduce the error.

Update of the NN

Lastly the *update()* function updates the weights for a given input of the network. For the training the cost function will consider the error between the desired state and the current state, but will not consider the steering angles.

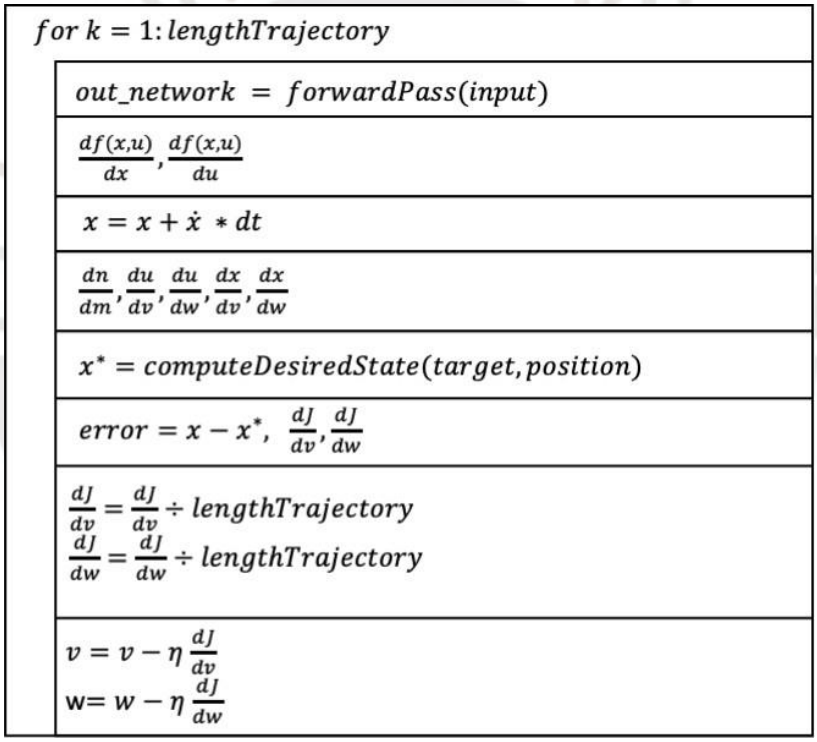


Figure 5.8: Structure chart of the update algorithm

In figure 5.8 one can see the structure of the algorithm. The equations have been developed in chapter 4.3.2. The update method is called batch and means that the mean of all derivatives of $\frac{dJ}{dv}$ and $\frac{dJ}{dw}$ over the whole trajectory is used to update the weights. After the whole trajectory is calculated the weights will be updated.

5.4 Pathplanning

Using the neuro controller for the model, various trajectories can be followed. To compute the desired state an algorithm has to be found. The algorithm distinguishes between the different target trajectories. For pathplanning of a straight line the *line of sight* methodology will be used, but for a circle the *perpendicular desired position* methodology as in [41] will be used, which computes the proper values of $x^*, y^*, \Psi^*, \delta_F^*$ and δ_R^* of the desired state.
Therefore, a perpendicular line is drawn between the actual position (x, y) of the robot and the trajectory. The intersection point will be (x^*, y^*). Ψ^* will be calculated using the derivative of the function of the trajectory at the intersection point.

5.4.1 Upwards

The task of driving upwards is the most simple of the paths. It is clear that the x-coordinate of the robot does not change over time and will be a constant and pre-defined. Because of the dynamics it could be possible to follow the trajectory with steered wheels and also an orientation, which is not $\frac{\pi}{2}$. This is the case of diagonal steering. With counter wheel steering it is not possible and the steering angles δ_F and δ_R as well as the orientation Ψ have to be zero. The possible scenarios are displayed in figure 5.9.

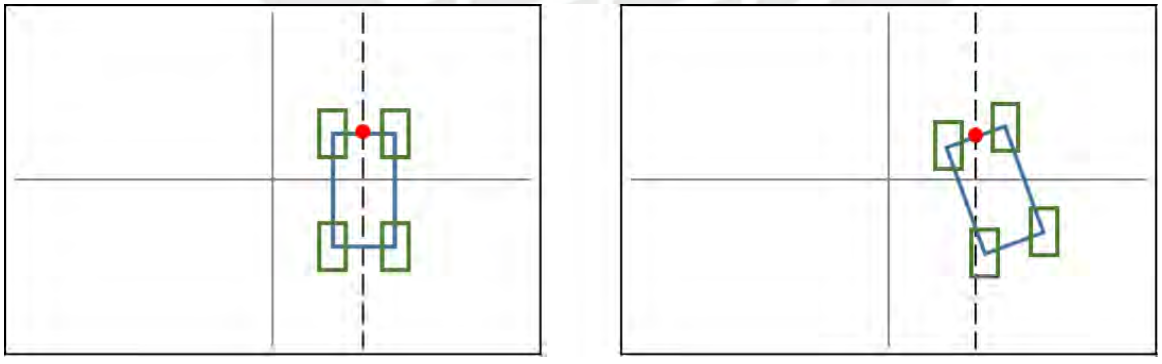


Figure 5.9: Desired states for upward driving

To simplify the solution the steering angles are zero in both cases and the orientation will be $\Psi = \frac{\pi}{2}$.

5.4.2 Straight line

For the pathplanning of the straight line another method will be used. The *line of sight* method uses one of the real coordinates of the robot to calculate the desired state. In figure 5.10, the y-coordinate of the current position is used also as the desired y-position.

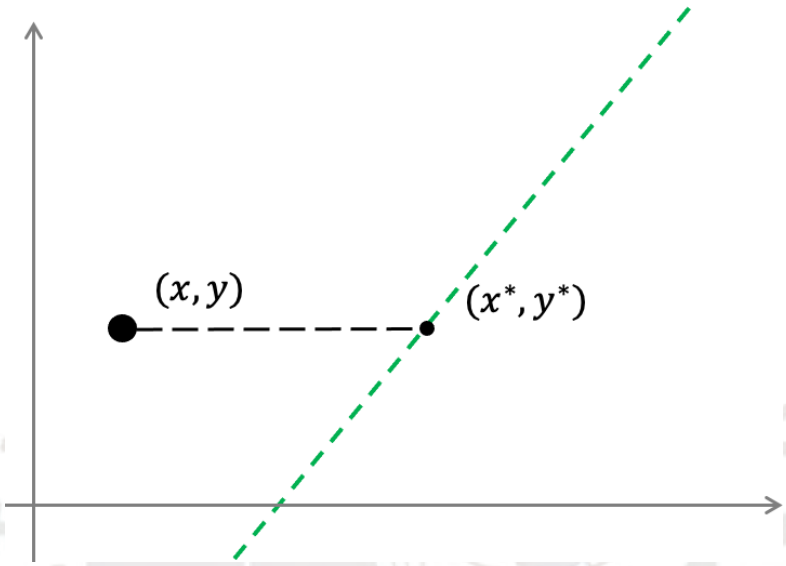


Figure 5.10: Computation of desired position for straight lines

To calculate the desired x-coordinate the equation for general straight functions will be used:

$$f(\bar{x}) = \bar{y} = m\bar{x} + n. \tag{5.2}$$

With a given $y^* = y$, x^* can be calculated with

$$x^* = \frac{y^* - n}{m}. \tag{5.3}$$

Like in figure 5.9, utilizing two cases can lead to a successful pathfollowing of a straight line. Also in this case only the case of non-steering will be considered. Therefore δ_F and δ_R are zero. By using the relationships shown in figure 5.11:

$$m = \frac{\Delta\bar{y}}{\Delta\bar{x}} = \tan(\Psi) \tag{5.4}$$

lead to

$$\Psi^* = \arctan(m). \tag{5.5}$$

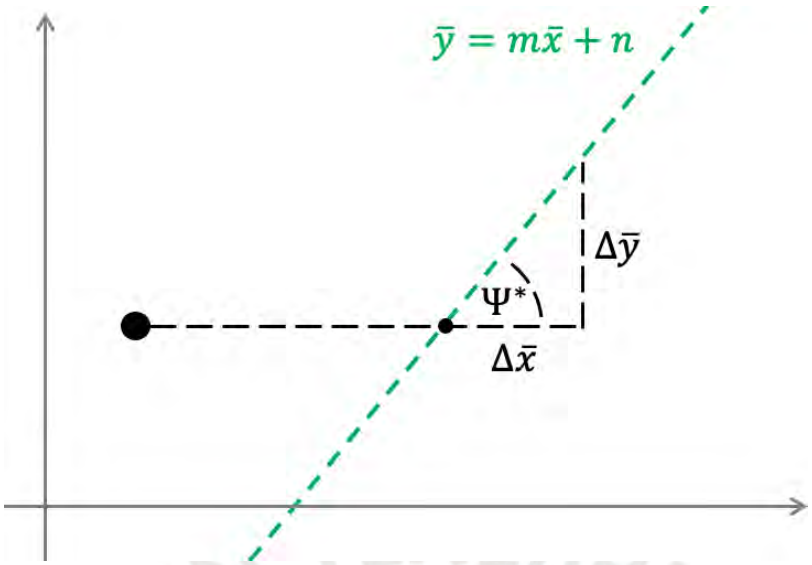


Figure 5.11: Geometrical relationships with *line of sight* method

5.4.3 Circle

The computation of the desired states of a circle-like path cannot be made using the line of sight method. This is because the circle can be positioned under or over the line of sight of the current position. So there is no intersection of the functions. That is why the *perpendicular desired position* methodology will be used.

A circle with radius R and center (x_c, y_c) has the equation

$$R^2 = (\hat{x} - x_c)^2 + (\hat{y} - y_c)^2 \tag{5.6}$$

for all $(\hat{x}, \hat{y})$ on the circle. This results in

$$\hat{y} = \pm \sqrt{R^2 - (\hat{x} - x_c)^2} + y_c \tag{5.7}$$

for $(x_c - R) \leq \hat{x} \leq (x_c + R)$ and with $\pm$ describing the upper and lower part of the circle, respectively.

A line with slope m and y-intercept n has the equation

$$\tilde{y} = m\tilde{x} + n \tag{5.8}$$

for all $x \in \mathrm{R}$.

The parameters m and n of the line connecting the robots current position (x, y) and

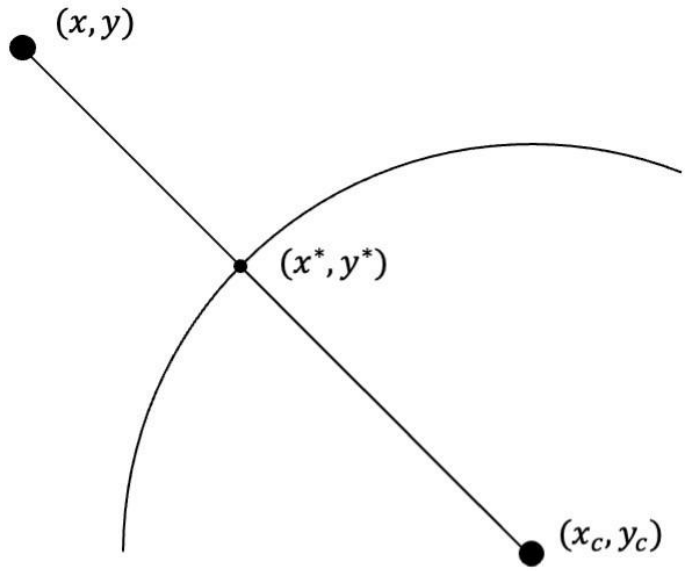


Figure 5.12: Computation of desired position for circular trajectories

the center of the circle (x_c, y_c) (assuming those points are different from each other) can be computed as follows:

$$m := \frac{(y - y_c)}{(x - x_c)}. \tag{5.9}$$

Using the known center point, one can obtain

$$y_c = \frac{(y - y_c)}{(x - x_c)} x_c + n \tag{5.10}$$

$$n = y_c - \frac{(y - y_c)}{(x - x_c)} x_c \tag{5.11}$$

$$\Rightarrow \tilde{y} = \frac{(y - y_c)}{(x - x_c)} \tilde{x} + y_c - \frac{(y - y_c)}{(x - x_c)} x_c . \tag{5.12}$$

As can be seen in Figure 5.12, this line intersects with the circle at some point (x^*, y^*), so the respective equations are equal at this point (assuming the intersection is with the upper part of the circle, lower part follows analogously):

$$\frac{(y - y_c)}{(x - x_c)} x^* + y_c - \frac{(y - y_c)}{(x - x_c)} x_c = \sqrt{R^2 - (x^* - x_c)^2} + y_c . \tag{5.13}$$

Canceling y_c and simplify yields:

$$\frac{(y - y_c)}{(x - x_c)}(x^* - x_c) = \sqrt{R^2 - (x^* - x_c)^2}. \tag{5.14}$$

Squaring both sides leads to:

$$\left(\frac{(y - y_c)}{(x - x_c)}\right)^2 (x^* - x_c)^2 = R^2 - (x^* - x_c)^2. \tag{5.15}$$

Now bring $(x^* - x_c)^2$ to the other side and simplify again:

$$\left(\frac{(y - y_c)}{(x - x_c)}\right)^2 + 1 \; (x^* - x_c)^2 = R^2. \tag{5.16}$$

Using the square root and defining $k := \left(\frac{(y-y_c)}{(x-x_c)}\right)^2 + 1$, the equations are reduced to

$$\sqrt{k}\,(x^* - x_c) = R \tag{5.17}$$

$$\Rightarrow \sqrt{k}\,x^* - \sqrt{k}\,x_c = R, \tag{5.18}$$

and finally

$$\Rightarrow x^* = \frac{R + \sqrt{k}\,x_c}{\sqrt{k}}. \tag{5.19}$$

Substituting k by its definition, pulling a factor out of the square root and canceling parts leads to the final result for x^*:

$$x^* = \frac{R + \sqrt{\left(\frac{(y-y_c)}{(x-x_c)}\right)^2 + 1}\; x_c}{\sqrt{\left(\frac{(y-y_c)}{(x-x_c)}\right)^2 + 1}} \tag{5.20}$$

$$\Rightarrow x^* = \frac{R + \left(\frac{1}{x - x_c}\right)\sqrt{(y - y_c)^2 + (x - x_c)^2 x_c}}{\frac{1}{(x - x_c)}\sqrt{(y - y_c)^2 + (x - x_c)^2}} \tag{5.21}$$

$$\Rightarrow x^* = \frac{R(x - x_c) + \sqrt{(y - y_c)^2 + (x - x_c)^2 x_c}}{\sqrt{(y - y_c)^2 + (x - x_c)^2}} \tag{5.22}$$

$$\Rightarrow x^* = \frac{R(x - x_c)}{\sqrt{(y - y_c)^2 + (x - x_c)^2}} + x_c \tag{5.23}$$

Note that this is a closed form solution for the x-value of the closest point to the robots

position on the upper circle.
Using equation (5.7) the respective y-value y^* can be obtained:

$$y^* = \sqrt{R^2 - (x^* - x_c)^2} + y_c \,. \tag{5.24}$$

With x^* from equation (5.20)

$$y^* = \sqrt{R^2 - \left(\frac{R(x - x_c)}{\sqrt{(y - y_c)^2 + (x - x_c)^2}} + x_c - x_c\right)^2} + y_c \,. \tag{5.25}$$

Canceling $x_c - x_c$, defining $c := (y - y_c)^2 + (x - x_c)^2$ and computing the square:

$$y^* = \sqrt{R^2 - \left(\frac{R(x - x_c)}{\sqrt{c}}\right)^2} + y_c \tag{5.26}$$

$$\Rightarrow y^* = \sqrt{R^2 - \frac{R^2(x - x_c)^2}{c}} + y_c \tag{5.27}$$

Expand with c and simplify yields

$$y^* = \sqrt{\frac{cR^2}{c} - \frac{R^2(x - x_c)^2}{c}} + y_c \tag{5.28}$$

$$\Rightarrow y^* = \sqrt{\frac{cR^2 - R^2(x - x_c)^2}{c}} + y_c \tag{5.29}$$

$$\Rightarrow y^* = \sqrt{\frac{R^2(c - (x - x_c)^2)}{c}} + y_c \tag{5.30}$$

Substituting c by its definition, pulling R^2 out of the square root, applying the root on the denominator and canceling $(x - x_c)^2 - (x - x_c)^2$:

$$y^* = \frac{R\sqrt{(y - y_c)^2 + (x - x_c)^2 - (x - x_c)^2}}{\sqrt{(y - y_c)^2 + (x - x_c)^2}} \tag{5.31}$$

$$\Rightarrow y^* = \frac{R\sqrt{(y - y_c)^2}}{\sqrt{(y - y_c)^2 + (x - x_c)^2}} \tag{5.32}$$

Finally, the root cancels the square and the final expression for y^* is

$$y^* = \frac{R(y - y_c)}{\sqrt{(y - y_c)^2 + (x - x_c)^2}} \tag{5.33}$$

Note that for $x^* \in [x_c - R, x_c + R]$, it is required that $y^* \in [y_c - R, y_c + R]$. This is ensured, as the intersection is on the circle.

The desired angle of the robot is a tangent of the circle and therefore orthogonal to the line mentioned before. The resulting slope is

$$m_\perp := \frac{-1}{m} = \frac{x_c - x}{y - y_c} \tag{5.34}$$

leading to angle

$$\Psi = \arctan\left(\frac{x_c - x}{y - y_c}\right) \tag{5.35}$$

As the angles will be small for the used circles (small curvature), they will be set to zero for computational efficiency:

$$\delta_F^* := 0 \tag{5.36}$$

$$\delta_R^* := 0 \tag{5.37}$$

The exact calculation can be done with the help of the geometric relations between the circular trajectory and the robot shown in figure 5.13.

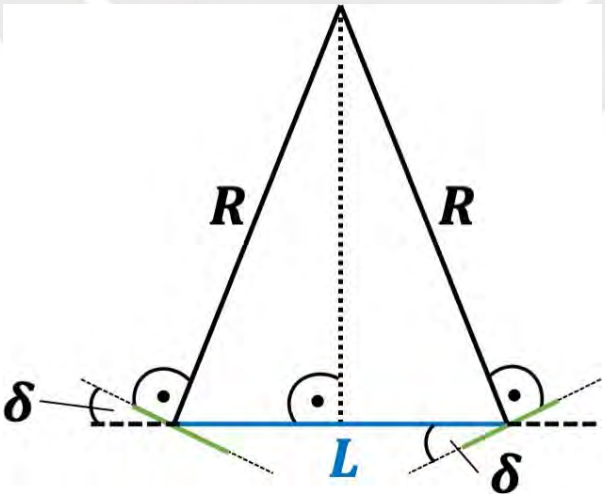


Figure 5.13: Geometrical relations for circular trajectories (1)

A section of the circle on which the robot is located is considered. The front axis and the rear axis are R far away from the center of the circle. In order to follow the circle, the wheels must be steered like tangents on the arc of the circle. This can be seen in the right angles drawn. It is easy to see that the robot can only move further along the circular line via counter wheel steering. Therefore, the steering angles must be the same at the front as at the rear.

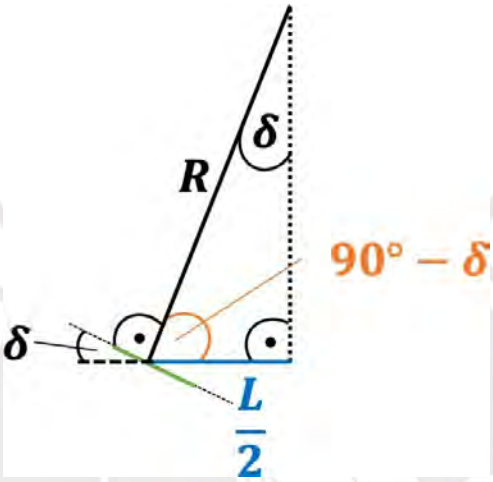


Figure 5.14: Geometrical relations for circular trajectories (2)

The triangle from figure 5.13 is symmetrical, so for further analysis only one side can be considered. In figure 5.14 the individual angles in the triangle are shown. The steering angle δ is present as an angle at intersected straight lines. So the lower left angle in the triangle can be described by $90^\circ - \delta$. Using the interior angle sum in the triangle, one can then calculate the upper angle, which is also δ. Finally, to calculate the angle δ, trigonometric equations are used. It follows:

$$\sin(\delta) = \frac{\frac{L}{2}}{R} = \frac{L}{2R} \tag{5.38}$$

$$\delta = \arcsin\left(\frac{L}{2R}\right) = \delta^*_F = \delta^*_R \tag{5.39}$$

$$\tag{5.40}$$

The simulations in Chapter 6 examine the difference between these two approaches.

ET LUX IN TENEBRIS LUCET
MCMXVII

6 Simulations

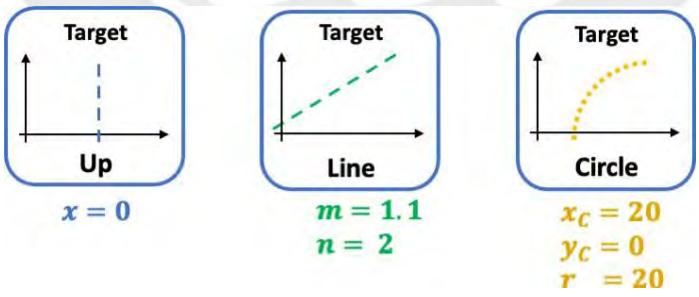
The system described in chapter 5 was used to simulate different scenarios. Therefore a selection of the parameters to be investigated must be made.

6.1 Simulation setup

To investigate the proposed controller and the overall system, a structured approach will be taken. Therefore, 12 cases are being considered. The general system uses a neural network with one hidden layer. The standard parameter set will be

width of the model	1.5 m	ϵ to target trajectory	2 m
velocity	1 m/s	length of trajectory	500
time step	0.1 s	number input neurons	2
δ_{max}	$\frac{\pi}{4}$	number of net trainings	5

The following describes the desired trajectories for the training process. In the simulations, deviating values were used to test what had been learned beyond the learning process.This also means how adaptable is what has been learned to new tasks.



The goal is to change one or more parameters to be able to draw a comparison. The following parameters are beeing considered:

- model with or without counter wheel steering,
- number of neurons in the hidden layer (20, 50, 100),

- activation function of the hidden layer (Sigmoid Type 1, Sigmoid Type 2, ReLu, Atan),
- number of orientations for incremental learning (2, 4, 8),
- learning rate η (0.1, 0.01, 0.001),
- length of the robot (2, 5),
- noise (on, off),
- target trajectory (up, line, circle) and
- number of learning epochs (100, 500, 1000).

6.2 Quality of performance

To estimate the quality of the algorithm, a region around the desired trajectory is defined. It should surround the trajectory like a tube. Disturbances can change the state of the robot. The tube indicates the area in which the robot should stay in the best case. The parameter, which describes the tube is *epsilon2target* and is in the standard parameter set set to two. In figures 5.4 and 5.5 one can see the area displayed in orange. The performance of the neuro controller is considered as good, if the robot will reach the tube in a given area.

6.3 Parameterization and incremental learning strategy

When training the neuro controller, challenges can arise due to incorrect parameterization. Such incorrect parameters lead to misbehavior. For training a standardized parameter set had to be able to draw comparisons later.
This training set is then used in section 6.4. In the following the standard behaviour of the training cases will be displayed. In figures 6.1 and 6.5 the different target trajectories are displayed. The model of counter wheel steering is used for the simulations.

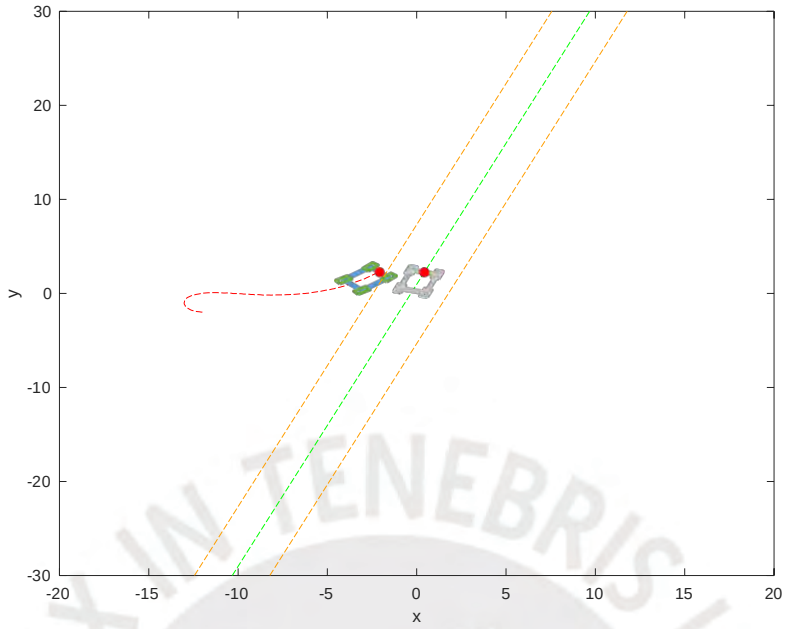


Figure 6.1: Target: Line (1)

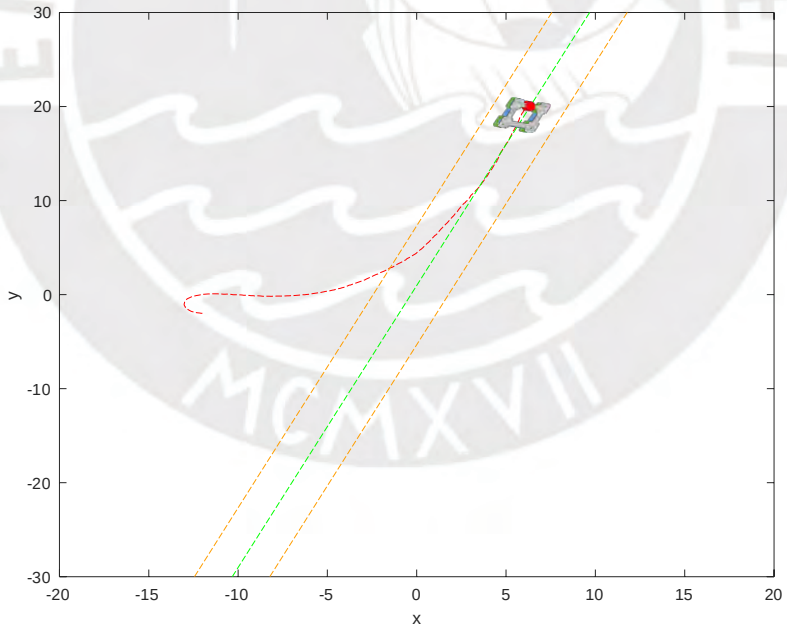


Figure 6.2: Target: Line (2)

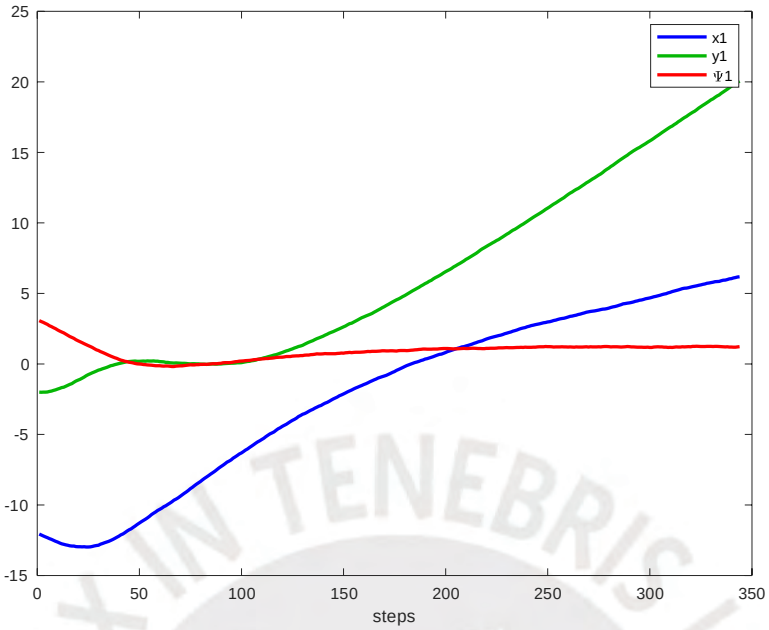


Figure 6.3: Target: Line (3)

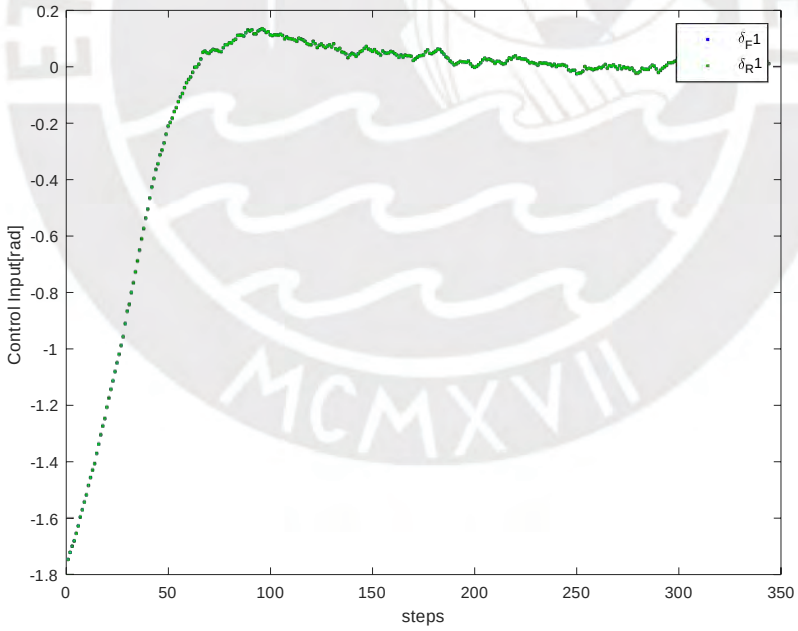


Figure 6.4: Target: Line (4)

There are two ways to define the desired steering angle for the circle trajectory. These possibilities were explained in chapter 5. In the following, the approaches are examined by simulation. In figure 6.5 the behaviour of the two models with counter wheel and independent steering is shown. With a desired steering angle of $\delta_F^* = \delta_R^* = 0$ the robot can not follow the trajectory for a long time.

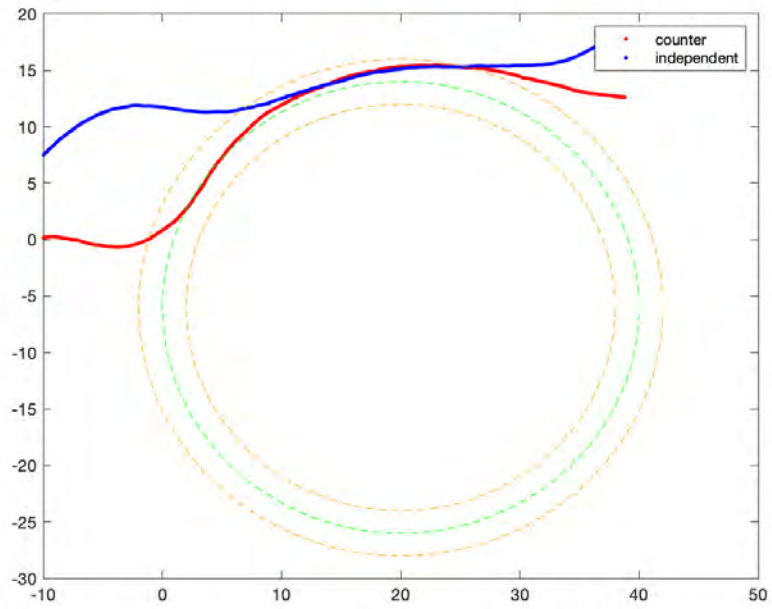


Figure 6.5: Target: Circle $\delta_F^* = \delta_R^* = 0$ (1)

In figure 6.6 it can be observed, that the driven paths are closer to the desired circular trajectory. There has been an improvement in the path following over a bigger number of steps. Both models follow the trajectory more closely. It can be observed, that the robot tends to leave the desired path after circa 500 steps. Which can be examined in figure 6.7. Although the desired state was adjusted to all circle sections.

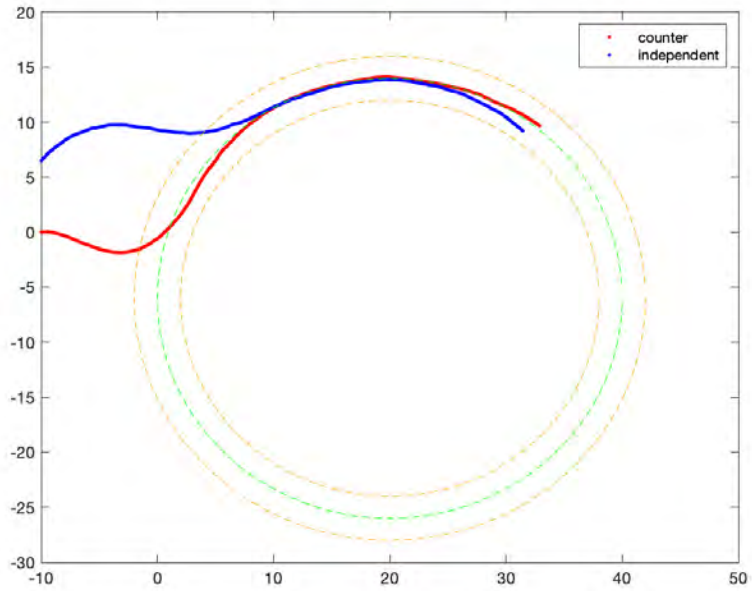


Figure 6.6: Target: Circle (2)

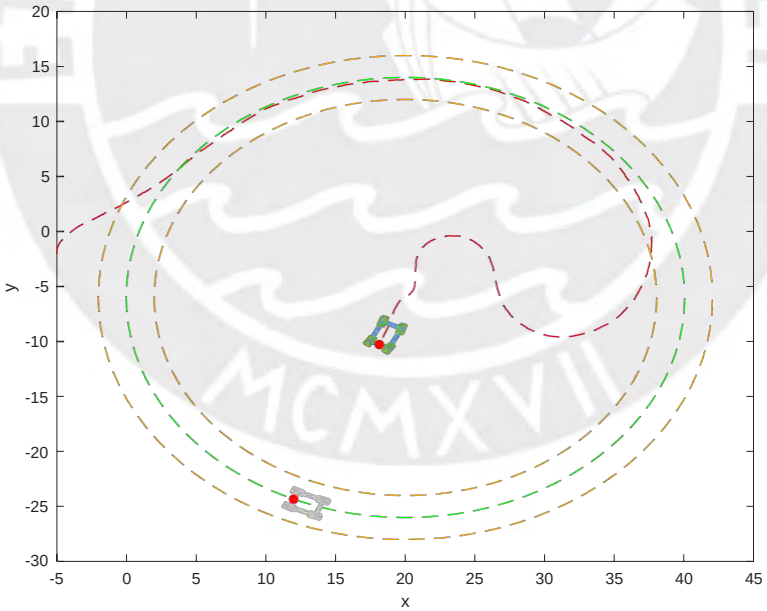


Figure 6.7: Target: Circle (3)

Various simulations have shown that it is a challenge to follow the entire circle, since a jump in the angle of orientation happens when going around. Because of its better performance the desired state is calculated exactly in the following simulations.

Influence of incremental points

Before examining the influence of the number of orientations at the incremental points in the next section, the initial points far away from the desired trajectory are compared to investigate the different reactions of the NN. For this purpose, training was performed with different numbers of incremental points. Therefore one distance means two incremental points at the coordinates $(-distance, 0, \Psi)$ and $(-distance, 0, \Psi)$. While the angle Ψ had also two different orientations $\frac{\pi}{2}$ and $\frac{3\pi}{2}$

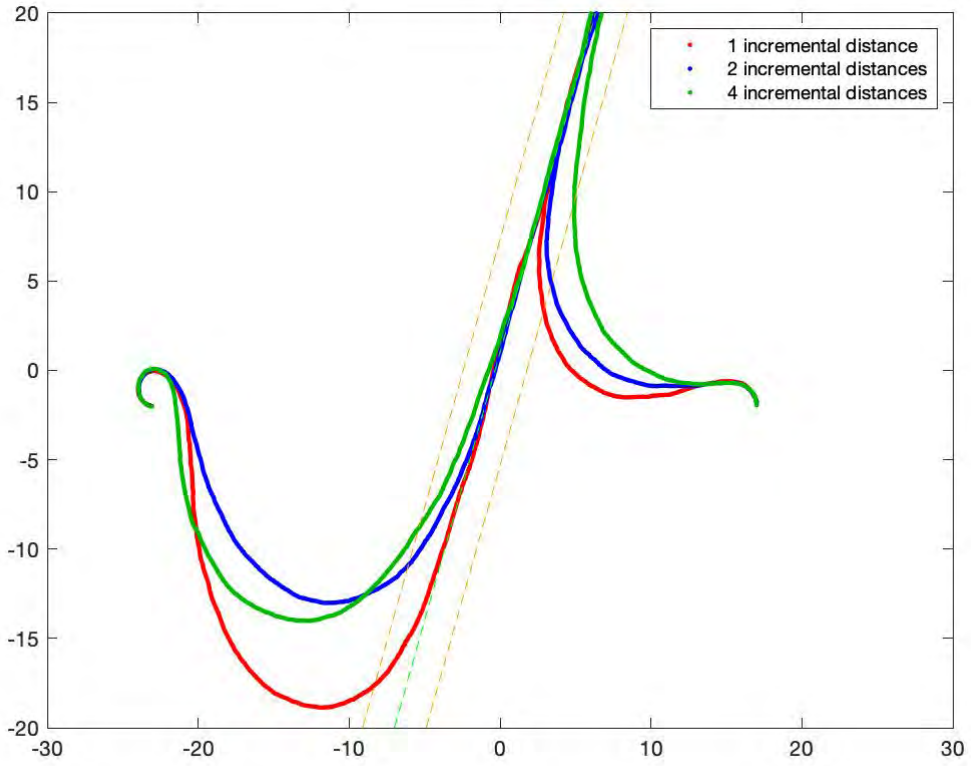


Figure 6.8: Behaviour of the networks with a different number of trained incremental points for line

All of the networks studied reached the target. Differences can be seen in the graphs. While the less trained network covers a longer distance than the other networks, the overall behavior is very similar. For the case of the initial point at $\begin{pmatrix} x \\ y \\ \Psi \end{pmatrix} = \begin{pmatrix} -23 \\ -2 \\ \pi \end{pmatrix}$, the robot initially steers toward the direction of the goal, however it over steers and then travels downward, away from the target, nevertheless it corrects itself and eventually

reaches the trajectory.

In figures 6.9 and 6.10 the target trajectory is switched to a circle. In the first step it appears that because of the complexity of the target, the initial points, which are at the limits, are closer to the y-axis. The networks reacted in different ways during the simulation. Two such simulations are shown in the figures 6.9 and 6.10. It can be observed, that with the circle trajectory there is not a similar or constant behavior. While in figure 6.9 the well trained network was only able to reach the target with an initial point with negative x coordinate, the red network, the less trained, follows the trajectory with an initial state with a positive x coordinate. In figure 6.10 the behaviours are very similar.

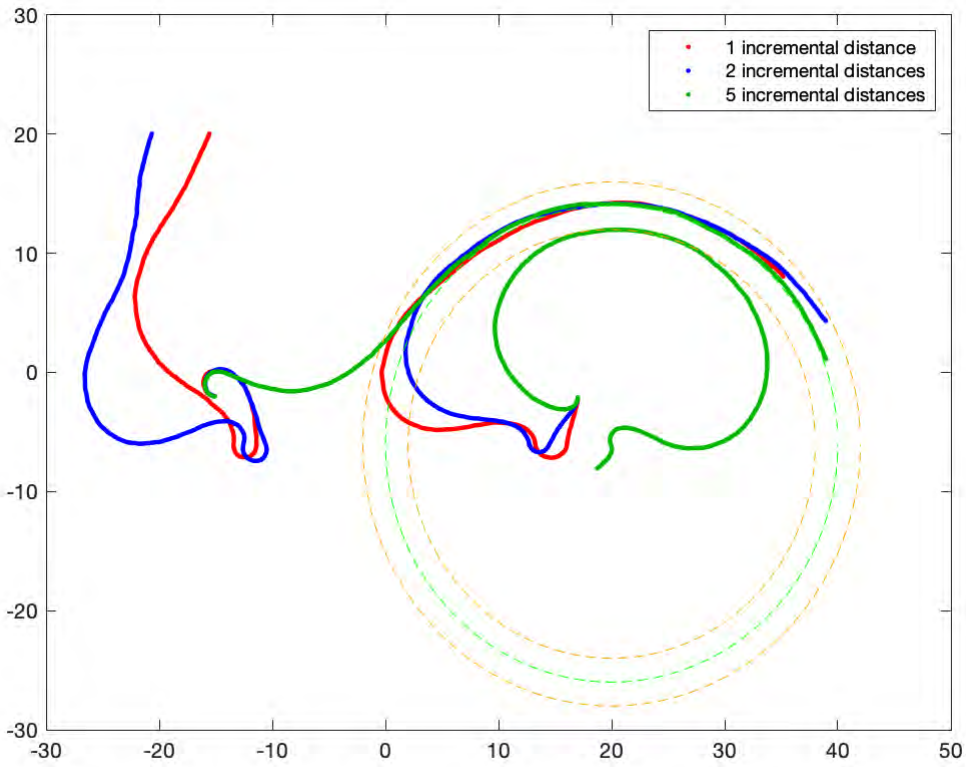


Figure 6.9: Behaviour of the networks with a different number of trained incremental points for circle (1)

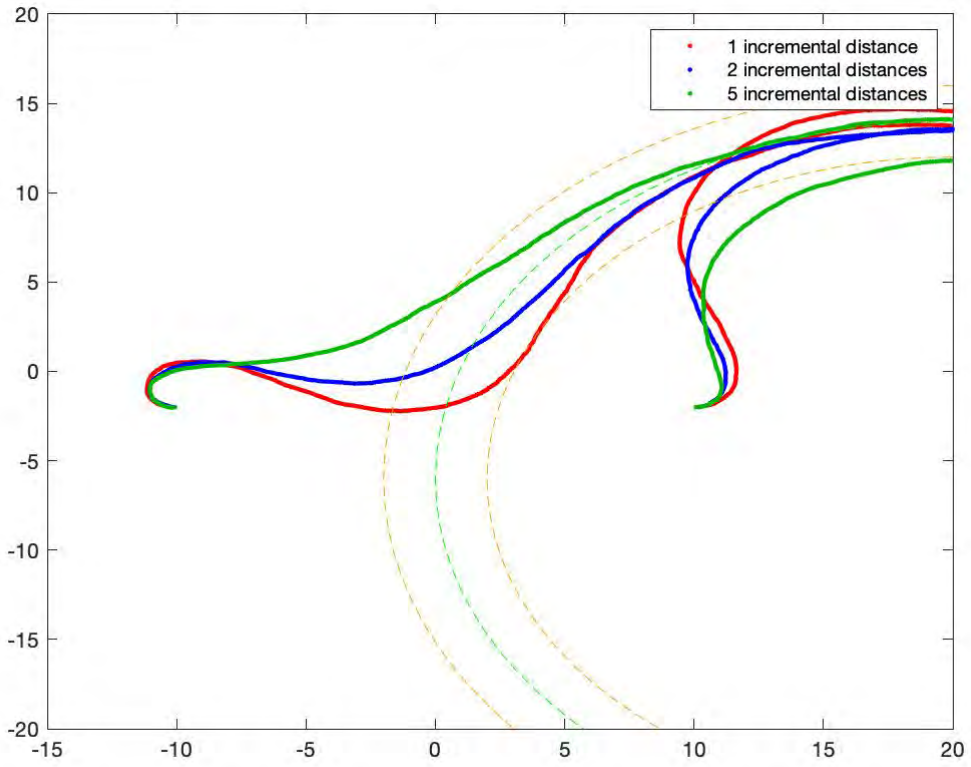
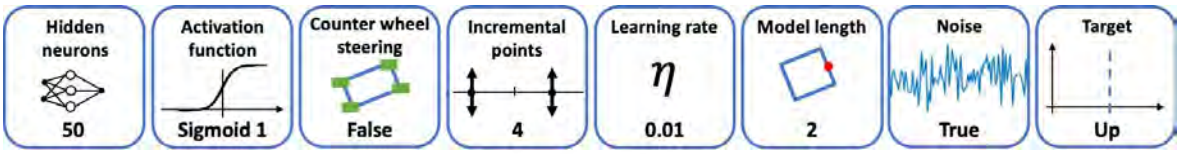


Figure 6.10: Behaviour of the networks with a different number of trained incremental points for circle (2)

6.4 Investigation of various parameters

In the following a set of parameters will be examined. Therefore an overview with the used parameters for this case will be given at the beginning. To consider the influence of model complexity, the general model is compared against counter wheel steering models.



At first the overall model behaviour will be introduced. Therefore the more complex model represented by equation 3.22 is used.

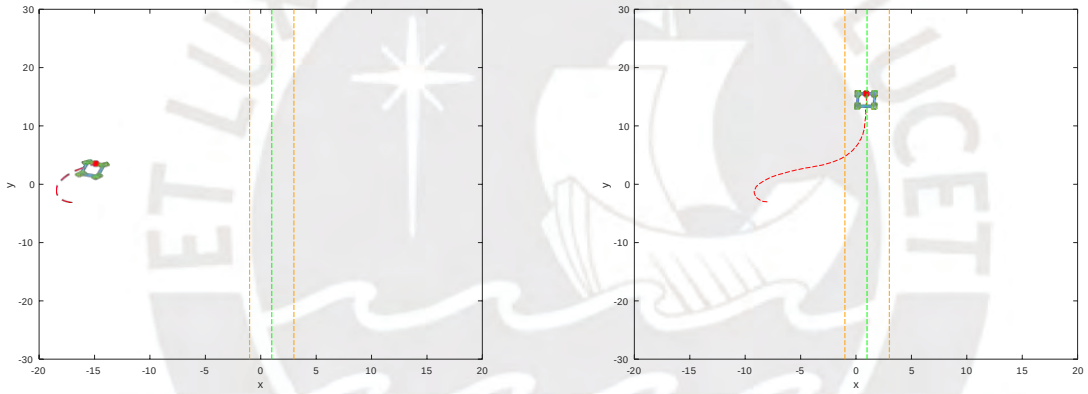


Figure 6.11: Independent steering and following the trajectory of case 1

In the figures 6.11 it is easy to see, that the robot is performing the task successfully. In sum, it begins with steering all wheels independently, then transitions to diagonal steering and then continues on the trajectory using only small steering angles, observed in 6.12.

The incremental distances were $[1, 2, 4]$, but the initial point is at $\begin{pmatrix} x \\ y \\ \Psi \end{pmatrix} = \begin{pmatrix} \cdot \\ -17 \\ \cdot \end{pmatrix}$. This shows how well trained the network is. In figures 6.12 it can be observed, that the desired states of $\begin{pmatrix} x^* \\ y^* \\ \Psi^* \end{pmatrix} = \begin{pmatrix} \cdot \\ \tilde{\cdot} \\ 0 \end{pmatrix}$ are reached. The steering angles are different and perform diagonal steering.

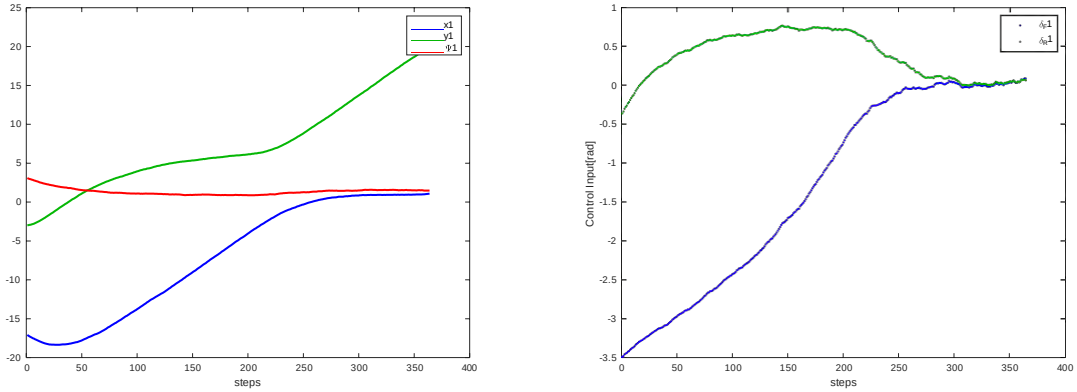
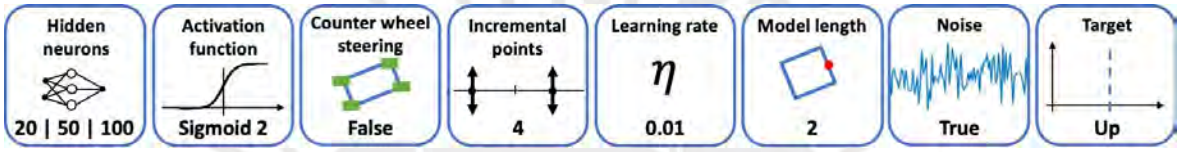


Figure 6.12: States and angles of case 1



In this case the number of hidden neurons is changed to study their influence. The networks were trained with the same parameters. To reduce the influence of the random initialization of the weights, five networks were trained and the best one was used for validation. The model used is the general model.

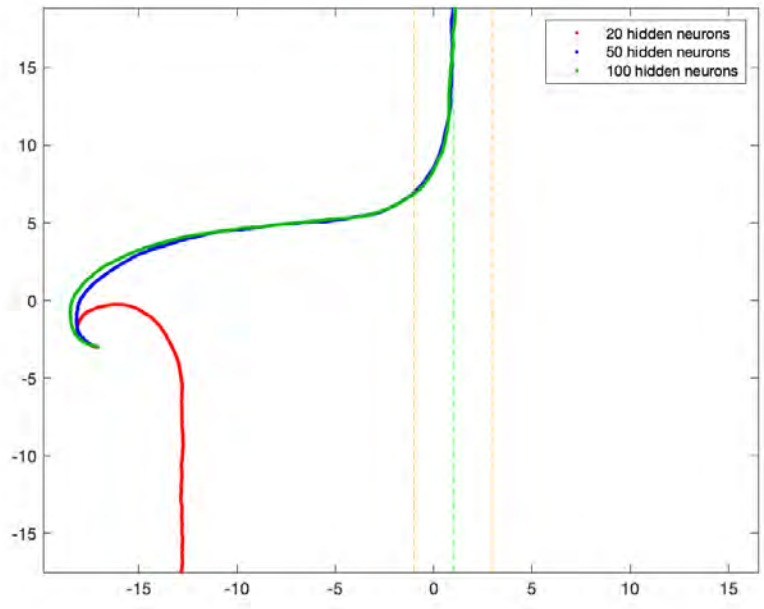
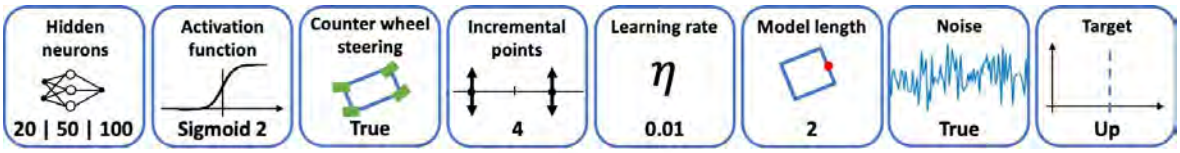


Figure 6.13: Behaviour of the different networks of case 2

The graphs in figure 6.13 show a case at the limits of the network. The initialisation point is far away from the learned points. The network with 20 hidden neurons in red does not succeed in performing the task. The networks with 50 or more hidden neurons perform very well. Furthermore, there is hardly any difference between the behaviors of the 50- and 100- hidden neuron networks.



The third case shows the same scenario with the less complex model. With counter wheel steering the network has to calculate just one steering angle, which will be used for the front and rear axis.

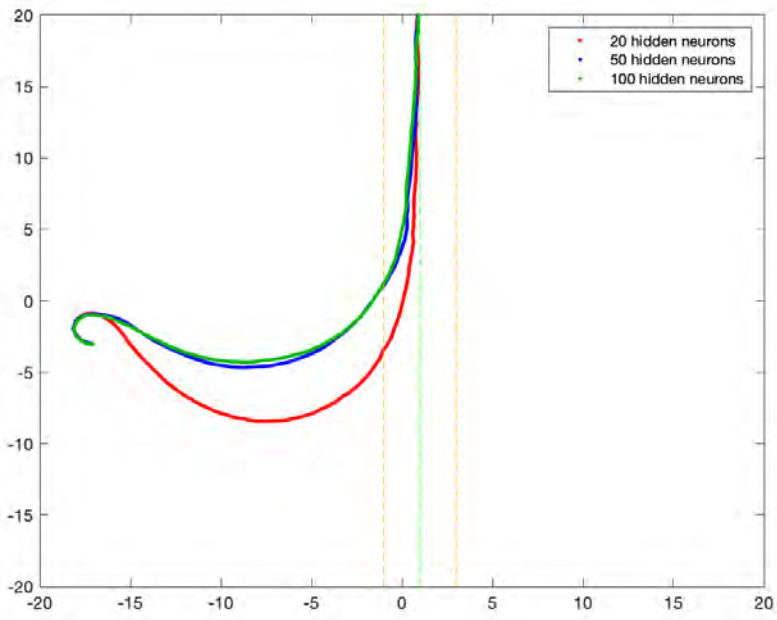
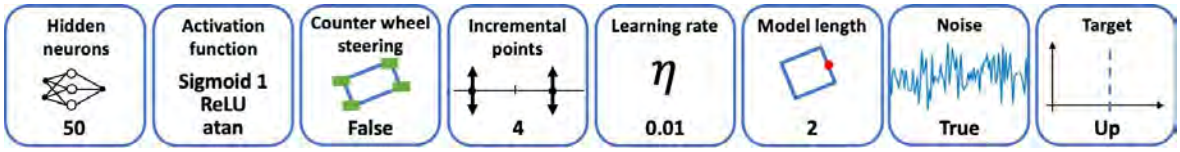


Figure 6.14: Behaviour of the different networks of case 3

In this case the network with 20 hidden neurons can complete the task. This shows, that the less complex model is able to perform the task with the same amount of training as the others.



The fourth case shows the dependency on the activation function of the hidden layer. While the network is trained in the standard training with a Sigmoid function type 2, the following figure 6.15 shows the behaviour with other activation functions. Similar to the other cases the initial point is at the boundary of the system.

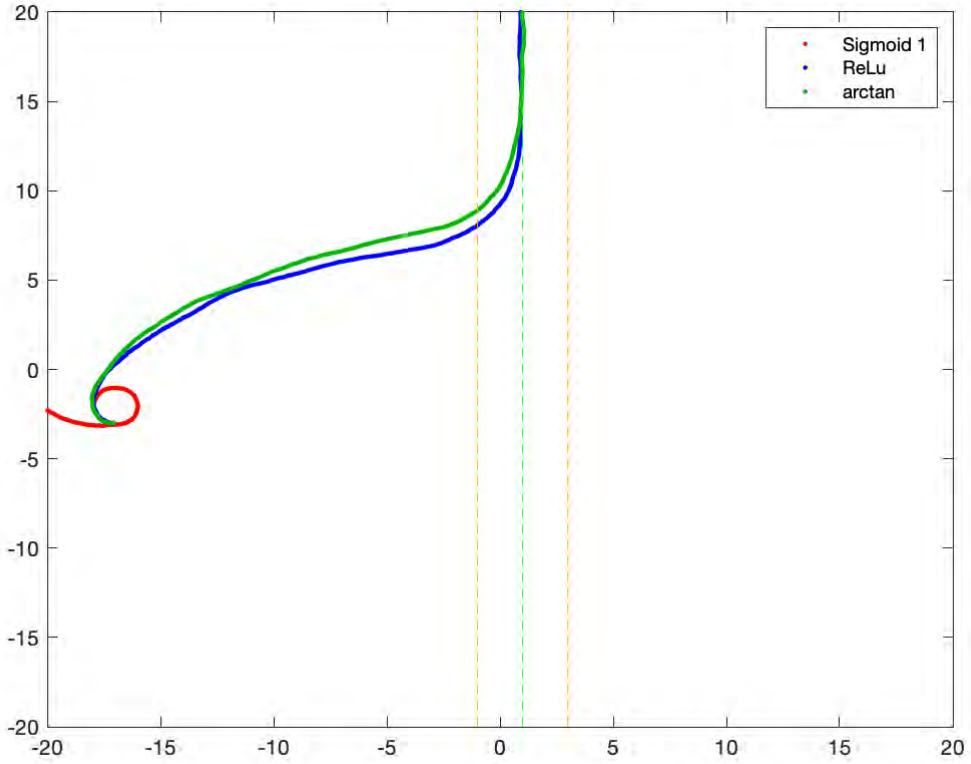


Figure 6.15: Behaviour of the different networks of case 4

The steering angles, displayed in figure 6.16, show the behaviour with the Sigmoid type 1 function. The calculated angles are out of any boundary and display the limit of this network. The other networks with ReLu and arctan as their activation function work with normal steering signals and are successful in path following.

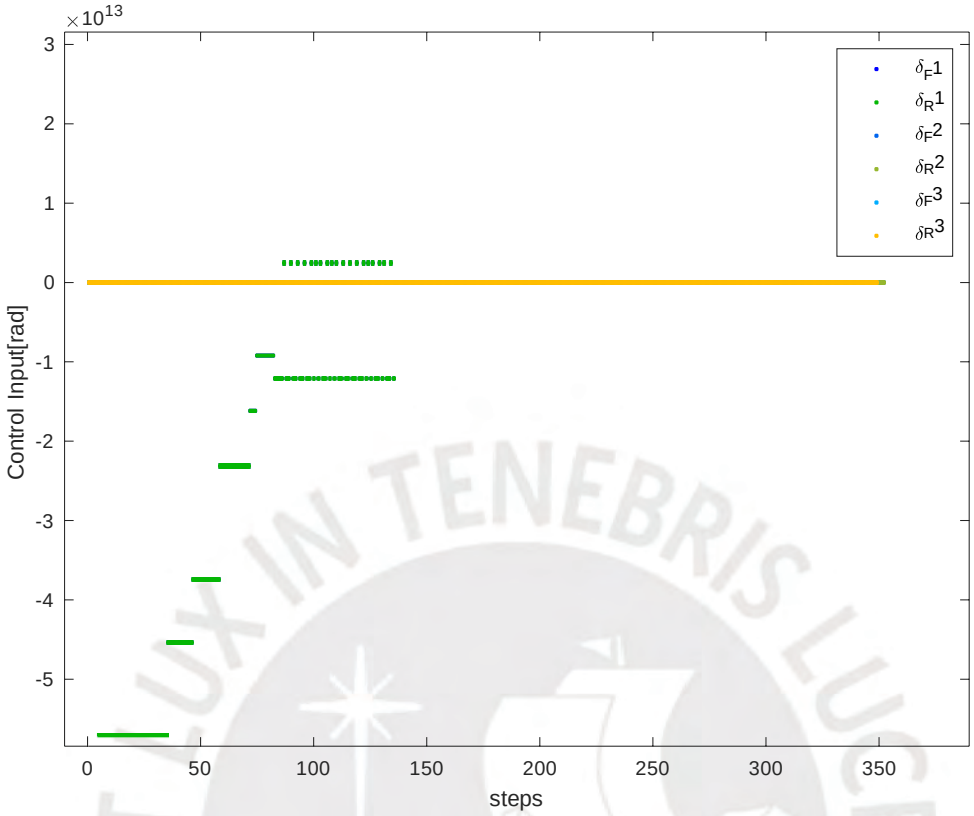
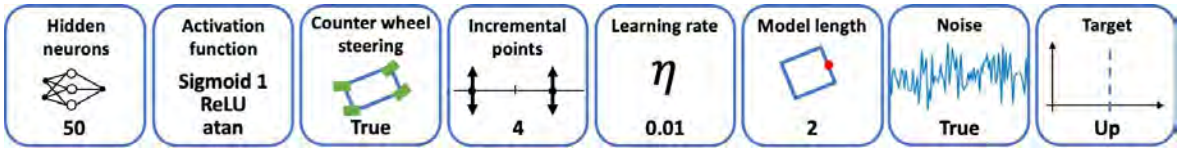


Figure 6.16: Steering angles of case 4



The fifth case holds the same as the fourth but again with the more simple model. The reduction in model complexity results in all networks performing the task well.

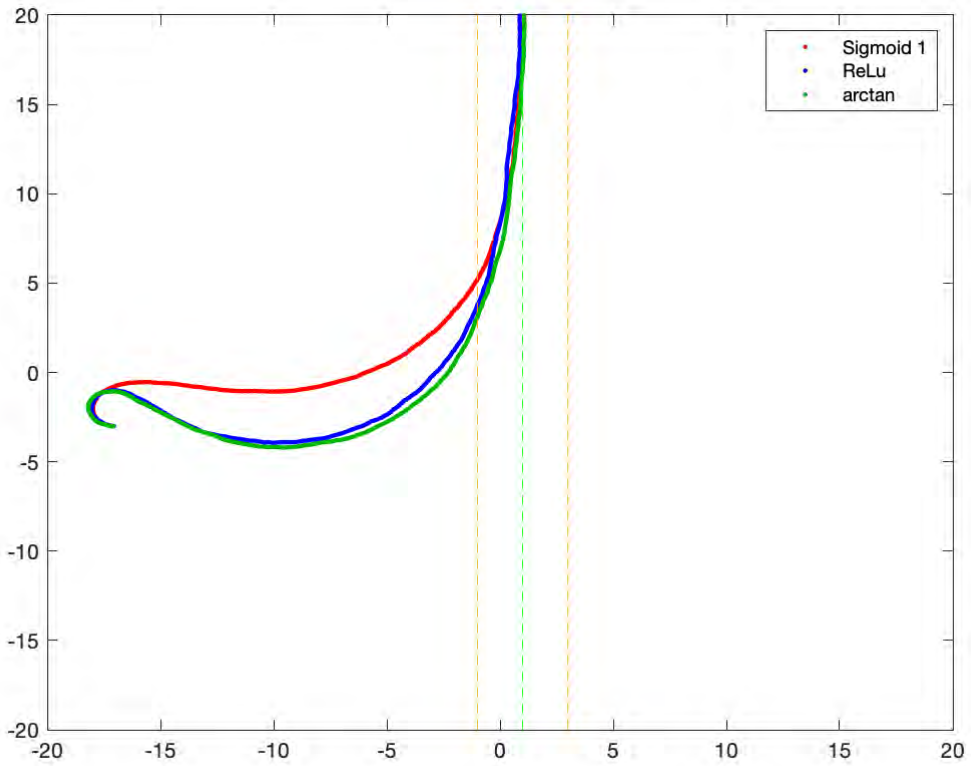
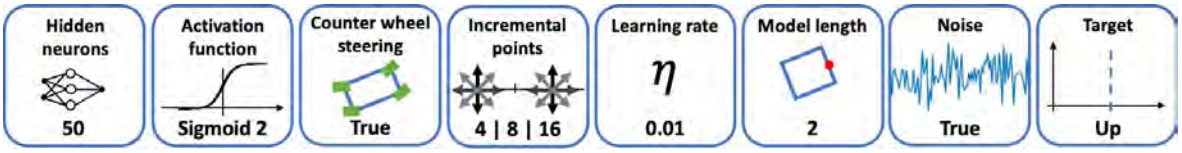


Figure 6.17: Behaviour of the different networks of case 5

The steering angles stay in the normal range of $\pm \frac{\pi}{4}$. Ultimately, the behaviour of the ReLu and Arctan functions are very similar. This shows that model complexity has to be considered while choosing an activation function.



A very interesting case is to investigate the influence of the number of initial points with different orientations. Therefore the set of distances for the incremental learning strategy is equal, but some networks have been trained with more initial points with more orientations. To analyze this behavior multiple initial points with different values of Ψ were simulated.

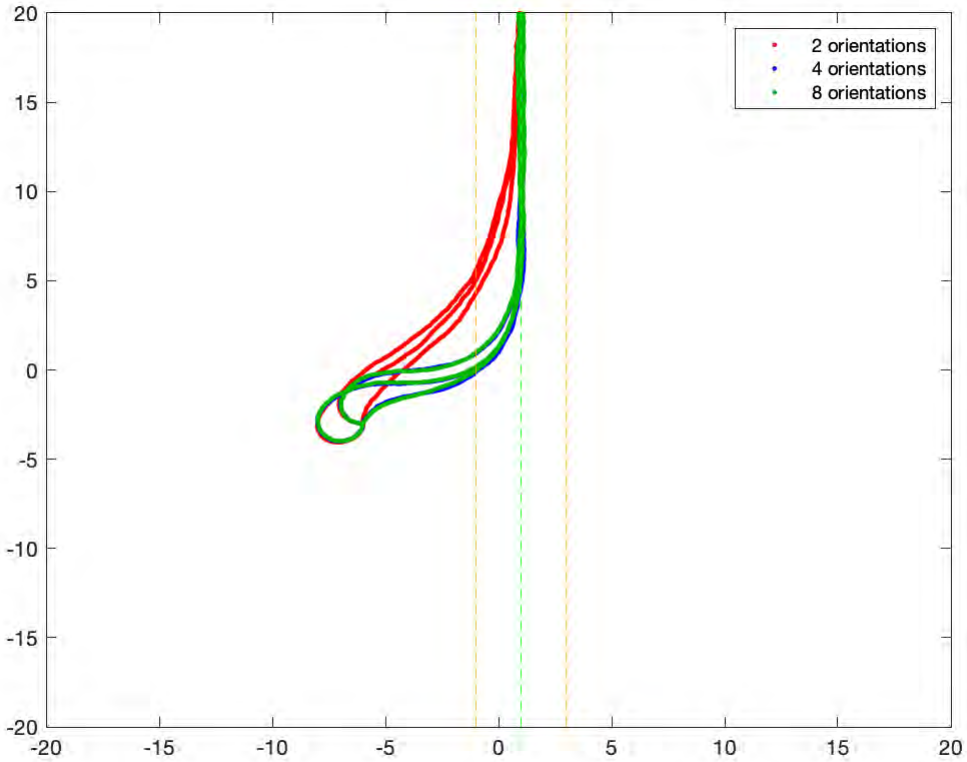
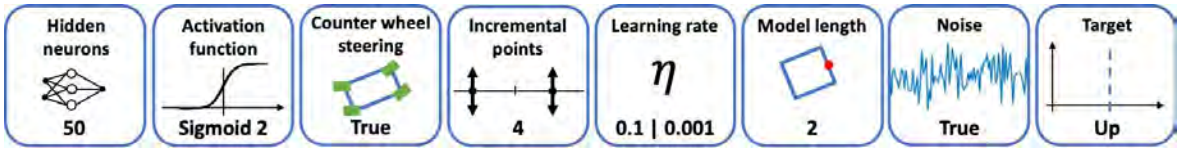


Figure 6.18: Behaviour of the different networks of case 6

There is a difference in the behaviour of the network trained with two different orientations at each point and the other two networks, trained with more orientations. The biggest difference is recognizable in the convergence to the trajectory. While the red graphs approach slowly, the others manage to get into the desired area faster.



The learning rate is a very powerful parameter. Often the learning process depends strongly on this parameter. In the seventh case two networks with a very high or low learning rate are being compared.

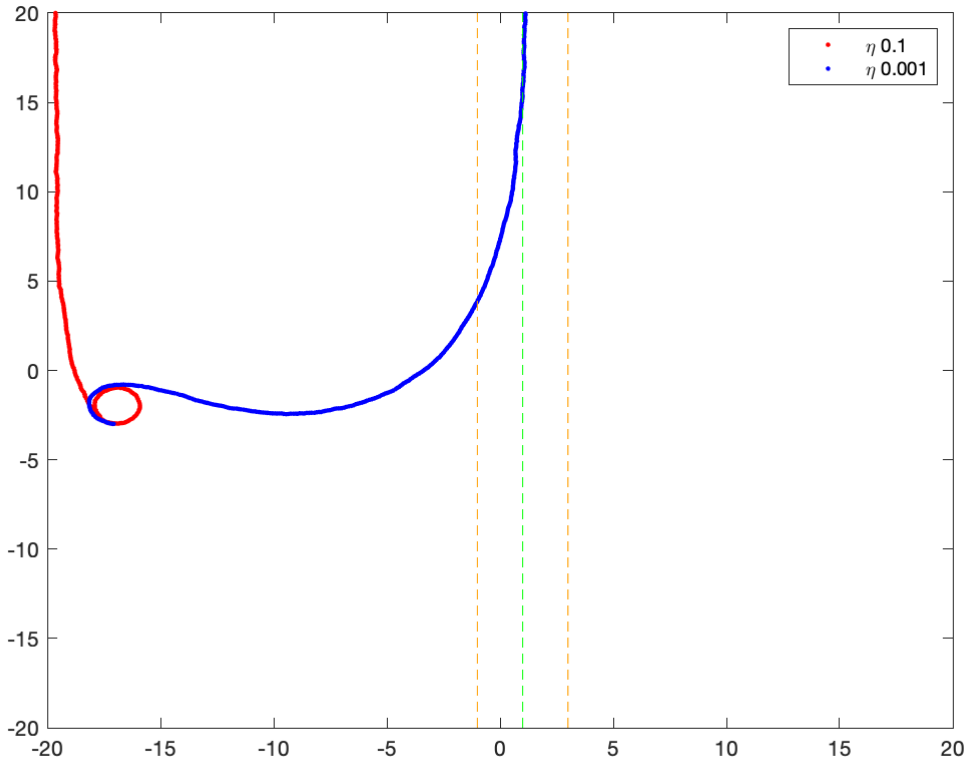
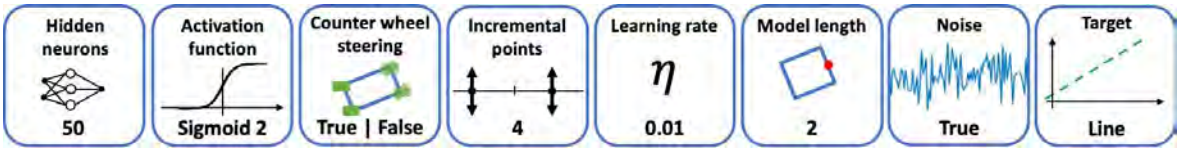


Figure 6.19: Behaviour of the different networks of case 7

The network with the large learning rate performs worse than the one with the very low learning rate. Both networks perform worse than the standard network with $\eta = 0.01$. An excessive learning rate leads to very large changes in the values of the weights, therefore they jump too much and no optimal values can be found.



Now the networks are also to perform more demanding tasks. For this purpose, a straight line is to be followed. The learned straight line has other parameters than the ones used for validation. Again the more complex model shall be compared with the simpler one.

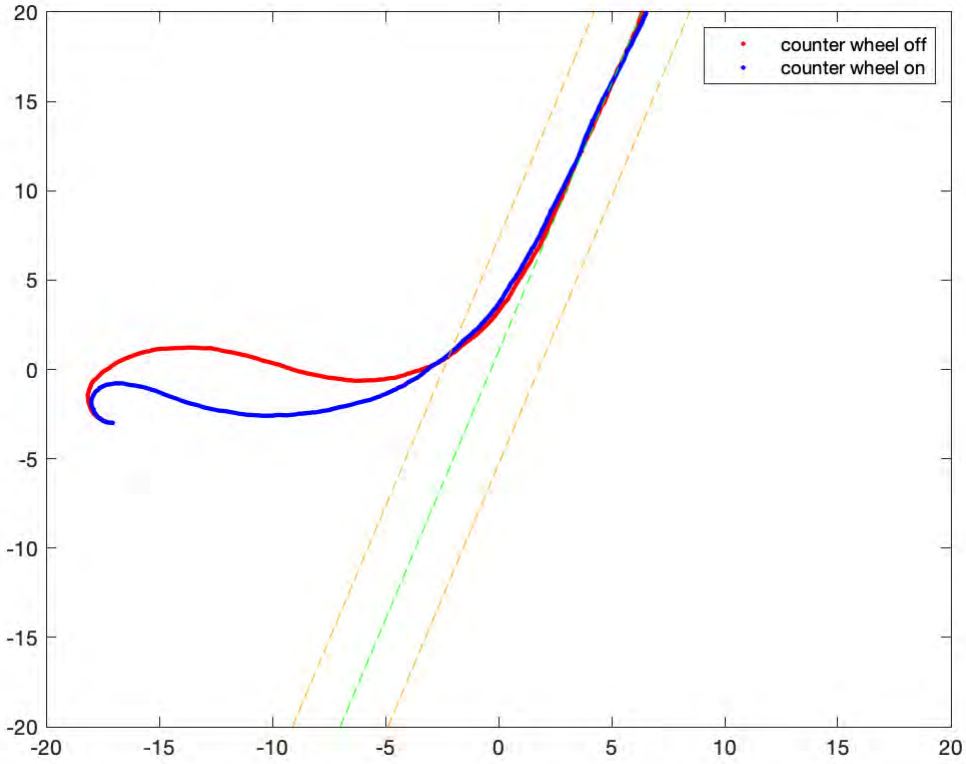
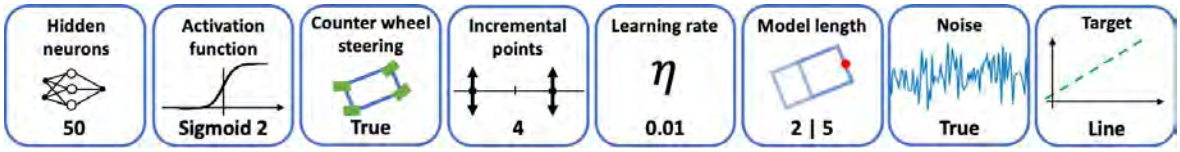


Figure 6.20: Behaviour of the different networks of case 8

The two networks are successful in completing the task. The distances traveled are different, but the entry point into the desired area is approximately the same.



The model dynamics depend strongly on the model parameters. The length of the robot is crucial for the performance.
During parameterization, it was noticed that a shorter robot is easier to train and can perform difficult tasks well due to its dynamics. Here we discuss the advantages of such dynamics.

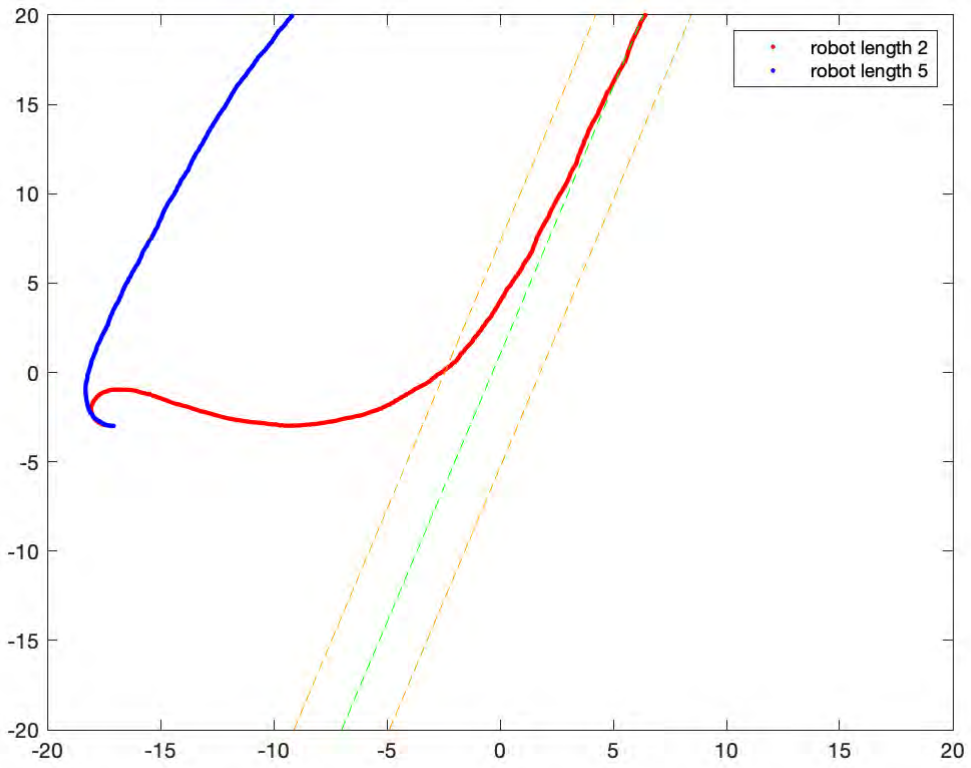
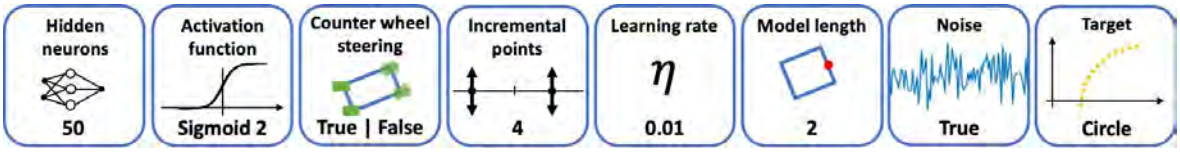


Figure 6.21: Behaviour of the different networks of case 9

In figure 6.21 one can see, how slowly the longer robot steers to get to the trajectory. Its dynamics inhibit the completion of the task.



To further complicate the trajectory, a circular path is now to be followed. Again, the two models are compared.

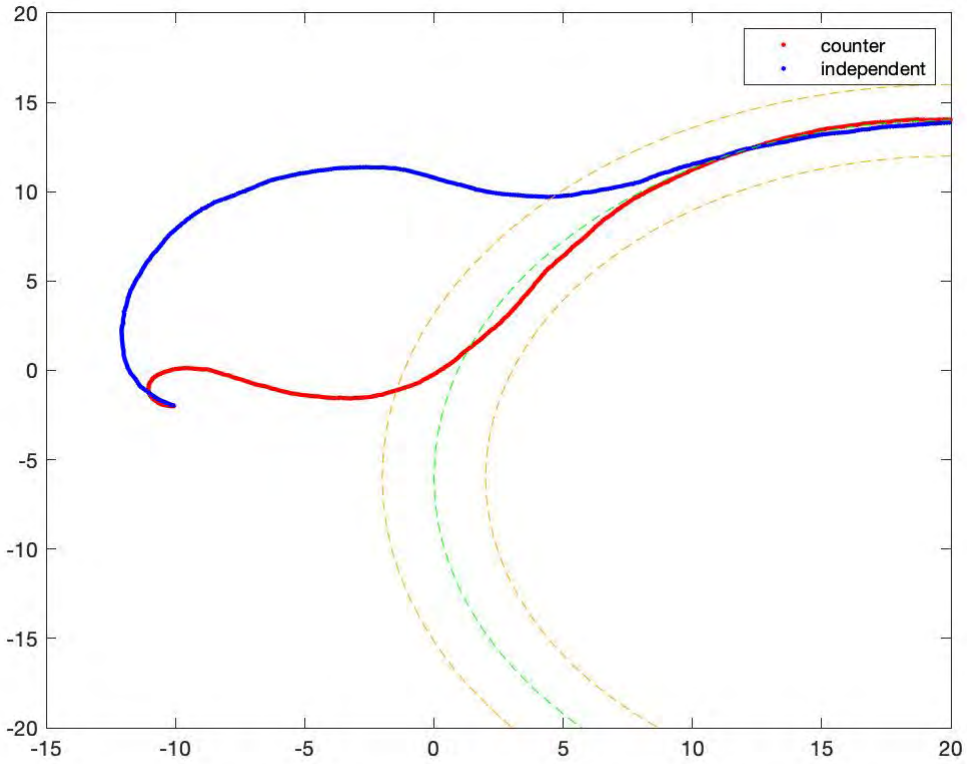
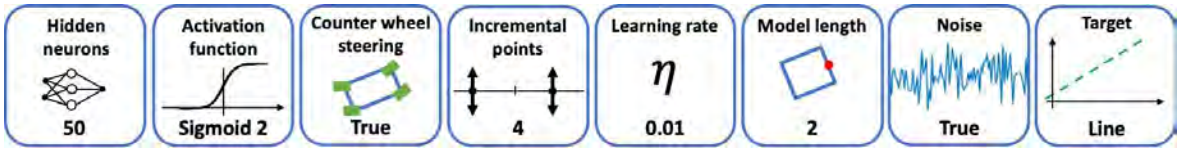


Figure 6.22: Behaviour of the different networks of case 10

For initial points closer to the circle both networks work well. But to display the limits of the networks a further point as well as a not learned orientation of $\Psi = \pi$ are chosen. The more complex model completes the task, but takes a longer path to reach the trajectory. In figure 6.22 one can see that both models reach the target trajectory by using completely different paths.



Lastly the influence of the epochs of training are to be examined. Therefore the target is to follow a line and in the next case a circular path.

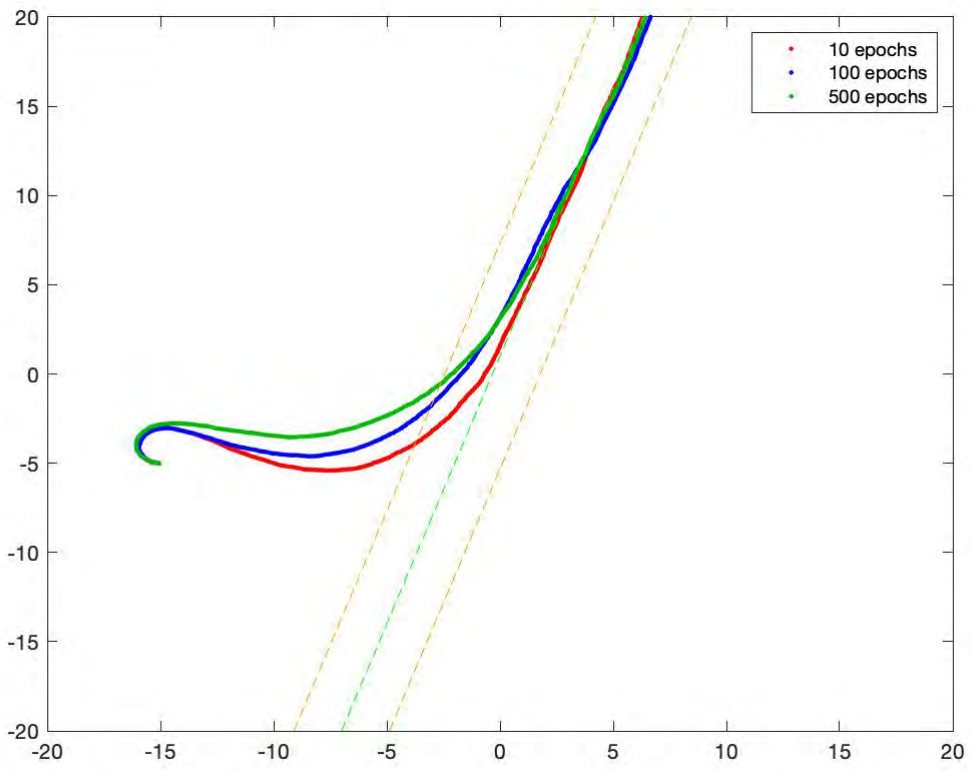
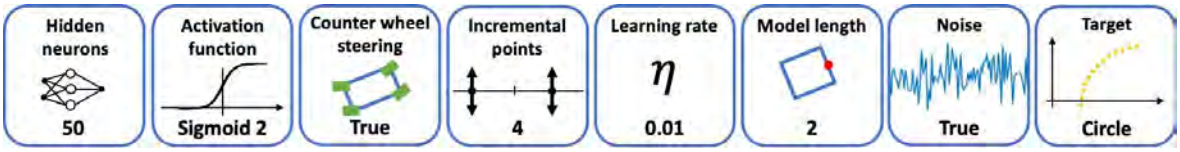


Figure 6.23: Behaviour of the different networks of case 11

In figure 6.23 it is obvious, that the networks generate different trajectories. The desired state will be found with the line of sight method. This means, that the best behaviour will be to go in a horizontal line to the desired line. This behaviour can be seen better the more epochs have been trained. The target is reached for all networks.



The more complex trajectory is the circle. In figures 6.24 and 6.25 the behaviours can be observed. While the less trained network does not achieve the goal, the more trained networks work well.

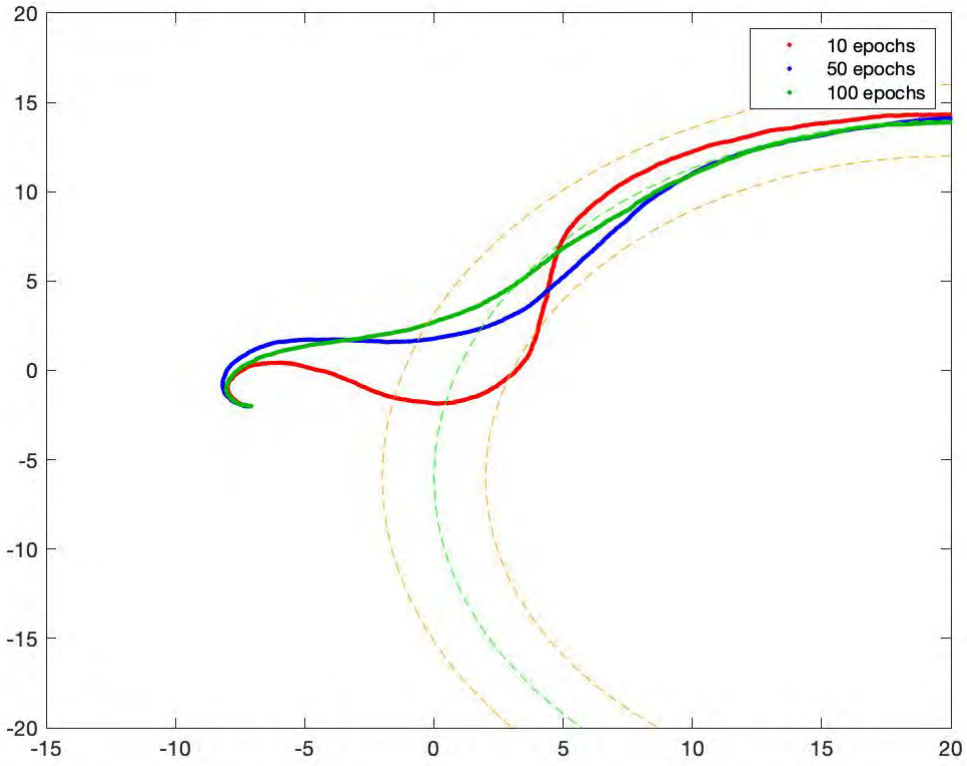


Figure 6.24: Behaviour of the different networks of case 11 (1)

This effect can be shown in greater distance to zero.It is remarkable that the network, which was trained with only one distance, still performs very well at distance of 7 and larger.

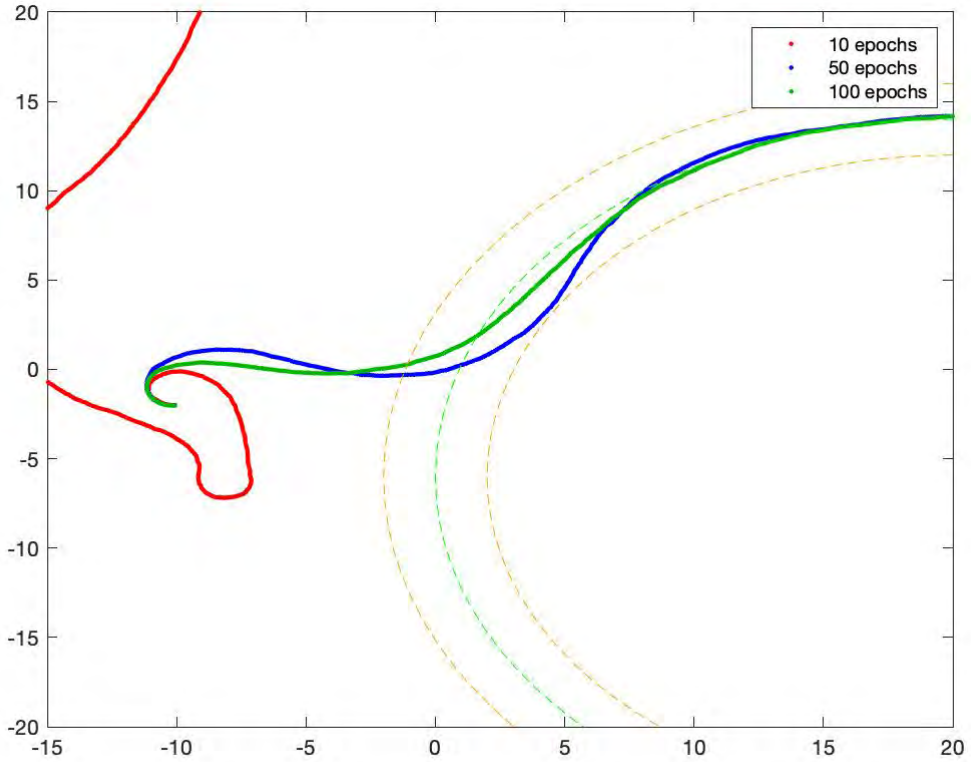


Figure 6.25: Behaviour of the different networks of case 11 (2)

6.5 Transferability to other tasks

In this section, it will be investigated whether the trained NN can also be applied to other target trajectories. The networks with the default parameters and the counter wheel steering model are selected.
The first network trained, for the path 'up', should perform a line trajectory. In figure 6.26 it can be observed, that the network can complete the task easily.

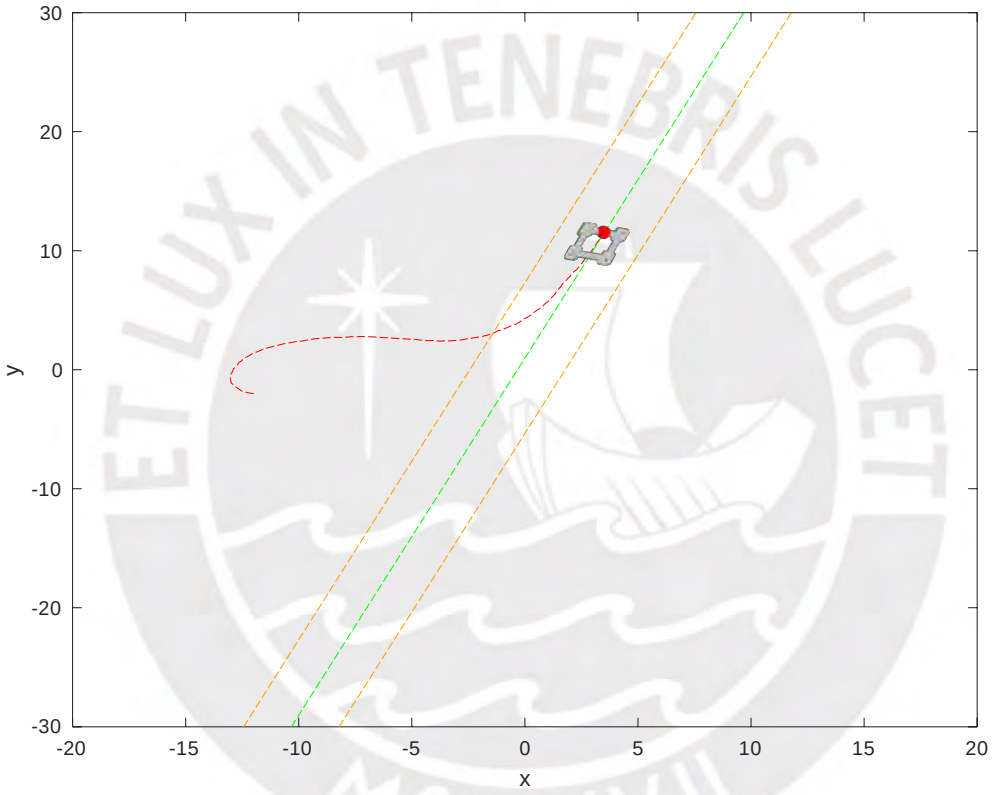


Figure 6.26: Evaluating the "up"-trained net with target type "line"

Now the network trained for "line" should perform a circle.Simulations were performed with a radius of 20 and 10 meters. The results are shown in the figures 6.27 and 6.28. The desired states of "line" and "circle" are similar to each other if the circle has a

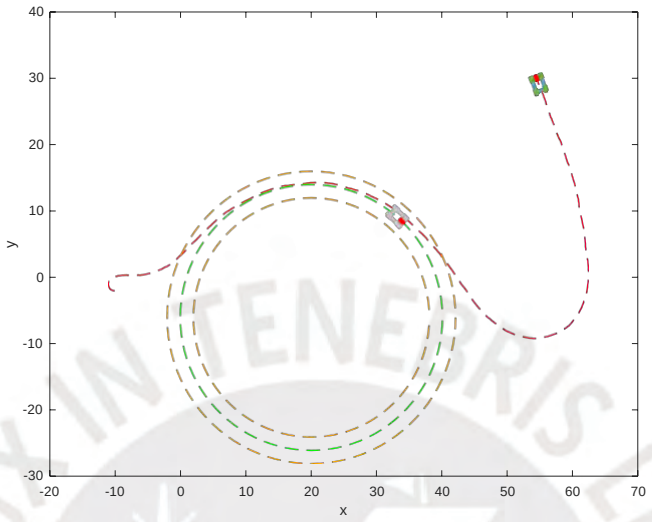


Figure 6.27: Evaluating the "line"-trained net with target type "circle"with $R = 20$

big radius so it can be considered as straight small areas, with steering angles equal to zero. So it can be interpreted as a straight line. The desired steering angles were calculated with the exact formula.

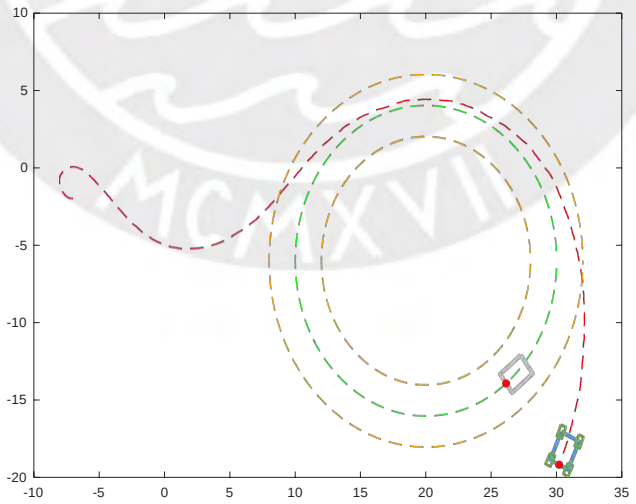


Figure 6.28: Evaluating the "line"-trained net with target type "circle" with $R = 10$

Thirdly, the network trained for circle paths should perform a line, illustrated in figure 6.29. Also this task is completed well.

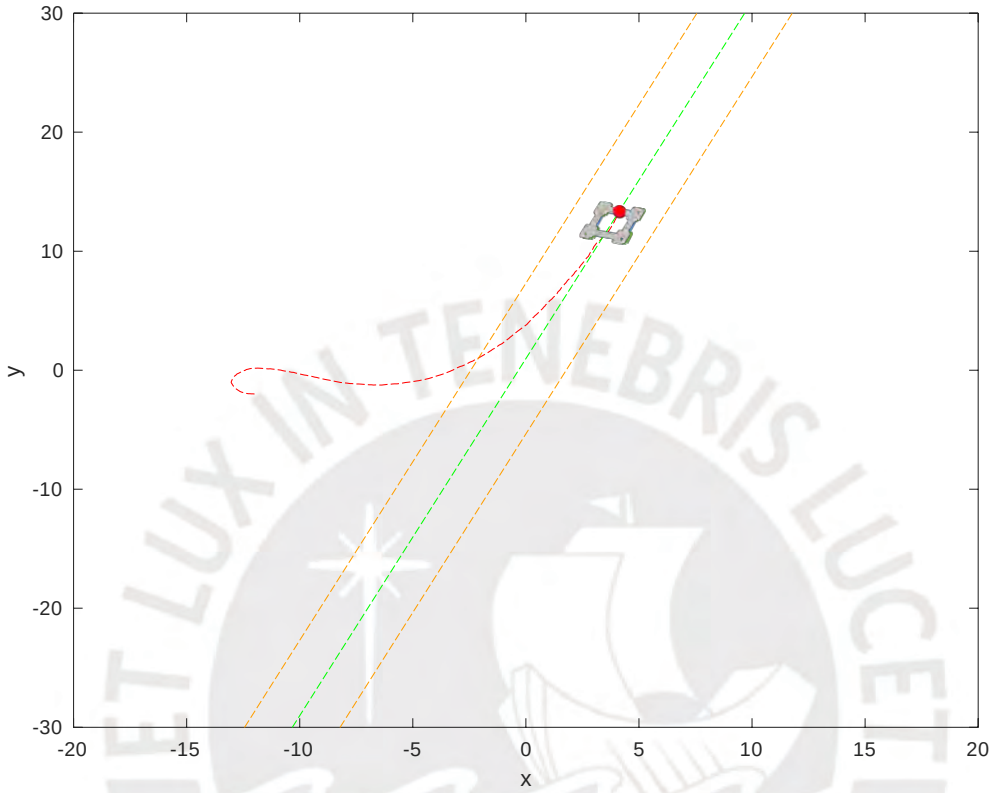


Figure 6.29: Evaluating the "circle"-trained net with target type "line"

Lastly, the network trained for the line trajectory is evaluated with different lines.

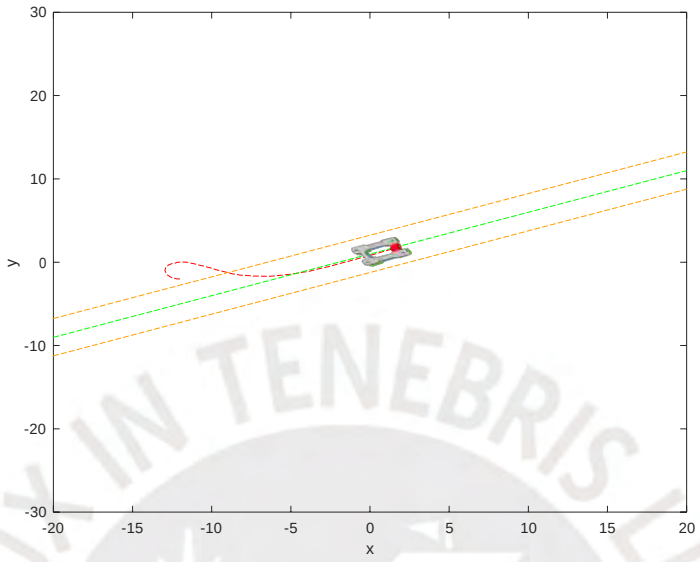


Figure 6.30: Evaluating the "line"-trained net with other targets type "line" (2)

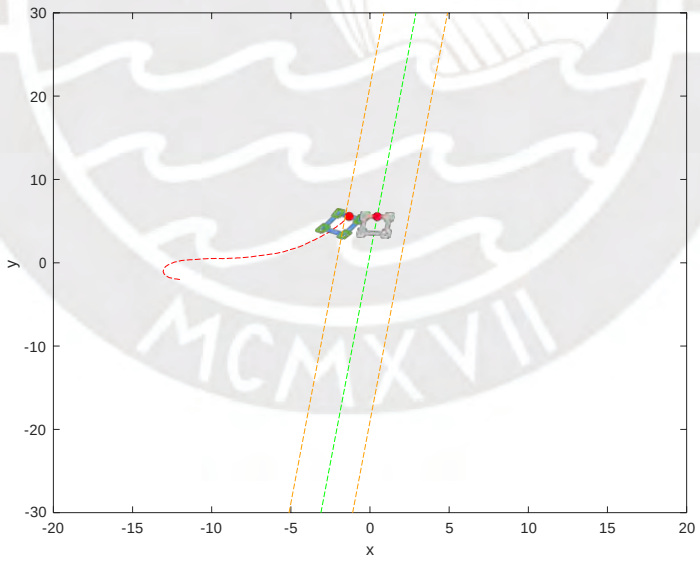


Figure 6.31: Evaluating the "line"-trained net with other targets type "line" (2)

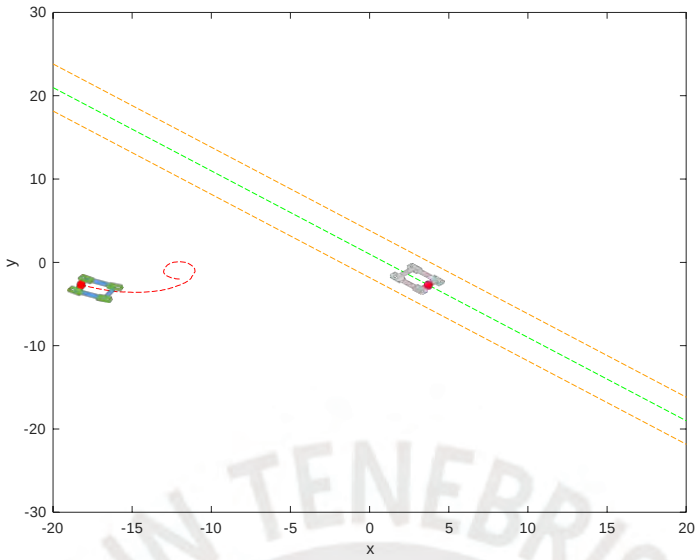


Figure 6.32: Evaluating the "line"-trained net with other targets type "line" (3)

As a summary of the figures 6.30, 6.31 and 6.32 it can be said that all straight lines that have a positive slope are well reached by the robot. Straight lines with negative slope cannot be reached. The robot moves in the wrong direction in this case.

6.6 Conclusions

After finding a parameter set which generates good solutions, many informative comparisons could be drawn.
The simulations were partly computed on a Workstation with Intel Core i7 6700 CPU (4 Cores @3.40 GHz) and 8GB of memory and on a 2017 MacBook Pro with Intel Core i5 CPU (2 Cores @3.1 GHz) and 16GB of memory. An average training with the standard parameter set takes approximately 5 minutes and 47 seconds. That means, that the computation time for all training situations was about 60 hours. Therefore the visualization was turned off to compute even faster.

The validations were made with initial states with a greater distance to the previous learned incremental points. It follows that the quality of the trained networks is very high, since almost all of them successfully completed the task. Only a few networks cannot perform the tasks to the same extent as others due to the intentionally inferior parameterization. Through validation, the differences between the more complex general model and the less complex counter wheel steering model become apparent.

The results show that even a very simple network structure with one hidden layer can lead to very good results. Nevertheless, parameterization and computing time play a major role.

The incremental learning strategy applies very well to the designed models. An improvement could be shown in various simulations, with different training sets.

ET LUX IN TENEBRIS LUCET
MCMXVII

7 Summary and outlook

For this work two different models of 4WS vehicles were introduced to perform trajectory following. The more general model allows independent steering angles on the front wheel and rear wheels. This leads to more flexibility and complexity. Counter wheel steering is a special and more simple case of the general model.

Furthermore a general neuro controller was described to control the nonlinear models. Therefore several networks have been trained to follow different trajectories. A standard parameter set was found and then used for further investigation of parameterization. Therefore several cases were examined with initial states at the limits of the trained network. Several simulations showed, that the approach of incremental learning leads to good performance with desired behavior even in areas that have not been trained. Every well trained net had successful task completion. Also the transferability was investigated and showed, that with the trained trajectory the network is able to perform other trajectories as well.

The introduced model has the advantage of generality. Through parameterization it is possible to simulate diagonal, front and counter wheel steering as well as independent steering. This allowed the complexity of the model to be changed at any point. Nevertheless, certain assumptions needed to be made about the model. The model was limited to two dimensions, and did not incorporate considerations for realistic side slipping, or nonlinear behavior of the wheels. Additionally, the system operates at a constant velocity.

Ultimately, the neuro controllers apply well to the system despite a short training process. Therefore the different kinds of trajectories were evaluated. The incremental learning strategy can increase the controllable area.

Outlook

Due to the results of the simulations, the learning strategy applies well to complex nonlinear models. Accordingly, nonlinear complex models are suitable for such a learning strategy. While the considered model is based on some assumptions, the next step is to include further influences like wheels and 3D behaviour. Beyond the model, the expansion of the path planning module is another development point. Complete trajectory planning in an environment with obstacles and limitations is an important part of autonomous driving.

A Appendix

The influence of noise

The mathematical model is a strong simplification of reality. However, in order to simulate sensor inaccuracies, a disturbance is introduced. The neural network is trained with this measurement disturbance, since such a disturbance will also exist in reality. This noise is implemented as an array of normally distributed random numbers. This array will be added to the calculated state after each step. In figure A.1 one can see the distribution of the values. Normal distributions have the advantage that more improbable values are represented. The amplitude of the noise effects the position as well as the state of the robot. Therefore, training with and without noise has been investigated in chapter 6.

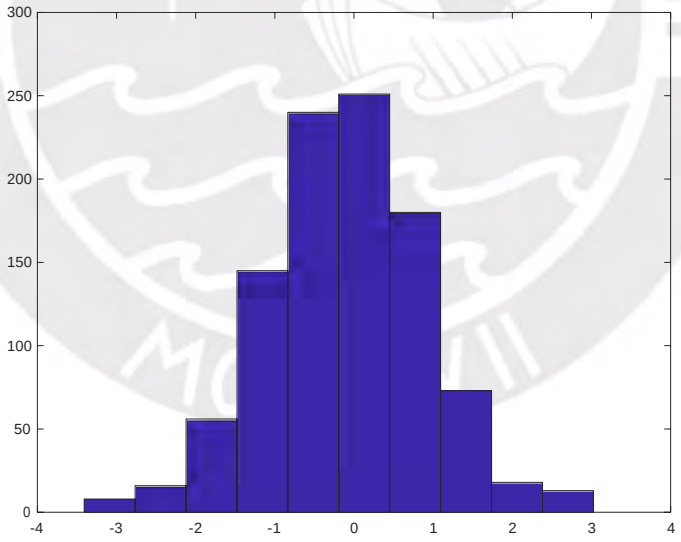


Figure A.1: Histogram of 1000 normally distributed random values with expected value 0 and standard deviation 1

ET LUX IN TENEBRIS LUCET
MCMXVII

Bibliography

[1] U. Meyer, *Die Kontroverse um Neuronale Netze: Zur sozialen Aushandlung der wissenschaftlichen Relevanz eines Forschungsansatzes*. Deutscher Universitätsverlag, 2013.

[2] EnjoyFirefighting International Emergency Response Videos, "All-radlenkung der Drehleiter + Ausrücken RTW BF München FW 5." https://www.youtube.com/watch?v=iQ4dW2HtW6c, accessed: 2020-07-05.

[3] H. Ihne, "Program of IAV 2016 - 9th IFAC Symposium on Intelligent Autonomous Vehicles in Leipzig, Germany."

[4] V. Müller, "Modellvalidierung und modell-prädiktiver Regelung von einem mobilen Roboter," Master's thesis, TU Ilmenau, 2013.

[5] A. Thieme, "Effizienzsteigerung modell-prädiktiver Regelungen und Anwendung auf einen mobilen Roboter," Master's thesis, TU Ilmenau, 2014.

[6] E. Drozdova, "Hinderniserkennung und -vermeidung durch einen mobilen Roboter im Rahmen einer modellprädiktiven Regelung," Master's thesis, TU Ilmenau, 2015.

[7] E. Belgradskaia, "Verbesserte Positionsbestimmung mittels Kalmann-Filter und neuronalem Netz bei der modellprädiktiven Regelung eines mobilen Roboters," Master's thesis, TU Ilmenau, 2016.

[8] J. P. Barreto Guerra, "Design of a Mobile Robot's Control System for Obstacle Identification and Avoidance using Sensor Fusion and Model Predictive Control," Master's thesis, PUCP Lima, 2017.

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

[10] Z. Rong-hui, C. Guo-ying, W. Guo-qiang, J. Hong-guang and C. Tao, "Robust Optimal Control Technology for Four-wheel Steering Vehicle," in *2007 IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 1513–1517.

[11] H. Yuan, Y. Gao, X. Dai and L. Yu, "Four-wheel-steering vehicle control via sliding mode strategy," in *2017 6th Data Driven Control and Learning Systems (DDCLS)*, pp. 572–577.

[12] S. Wu, E. Zhu, M. Qin, H. Ren and Z. Lei, "Control of Four-Wheel-Steering Vehicle Using GA Fuzzy Neural Network," in *2008 International Conference on Intelligent Computation Technology and Automation (ICICTA)*, pp. 869–873.

[13] S. Wang and J. Zhang, "The research and application of fuzzy control in four-wheel-steering vehicle," in *2010 7th International Conference on Fuzzy Systems and Knowledge Discovery*, pp. 1397–1401, 2010.

[14] L. Tan, S. Yu, Y. Guo and H. Chen, "Sliding-mode control of four wheel steering systems," in *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 1250–1255.

[15] N. Hamzah, M. K. Aripin, Y. M. Sam, H. Selamat and M. F. Ismail, "Yaw stability improvement for four-wheel active steering vehicle using sliding mode control," in *2012 IEEE 8th International Colloquium on Signal Processing and its Applications*, pp. 127–132.

[16] F. Du, J. Li, L. Li, Y. Xue, Y. Liu and X. Jia, "Optimum control for active steering of vehicle based on $H\infty$ model following technology," in *2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics (CAR)*, pp. 341–344.

[17] Y. Cao and M. Qiao, "Application of fuzzy control in four wheel steering control system," in *2017 International Conference on Advanced Mechatronic Systems (ICAMechS)*, pp. 62–66.

[18] L. Qiang, W. Huiyi and G. Konghui, "Identification and control of four-wheel-steering vehicles based on neural network," in *Proceedings of the IEEE International Vehicle Electronics Conference (IVEC)*, pp. 250–253, 1999.

[19] B. Dumitrascu, A. Filipescu, V. Minzu, A. Voda and E. Minca, "Discrete-time sliding-mode control of four driving-steering wheels autonomous vehicle," in *Proceedings of the 30th Chinese Control Conference*, pp. 3620–3625, 2011.

[20] B. Dumitrascu, A. Filipescu, C. Vasilache, E. Minca and A. Filipescu, "Discrete-time sliding-mode control of four driving/steering wheels mobile platform," in *2011 19th Mediterranean Conference on Control Automation (MED)*, pp. 1076–1081.

[21] A. Moran Cardenas, J. G. Rázuri, D. Sundgren and R. Rahmani, "Autonomous Motion of Mobile Robot Using Fuzzy-Neural Networks," in *2013 12th Mexican International Conference on Artificial Intelligence*, pp. 80–84.

[22] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[23] K. S. Narendra and K. Parthasarathy, "Gradient methods for the optimization of dynamical systems containing neural networks," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 252–262, 1991.

[24] S. F. Rezeka, "A feedforward neural network for the automatic control of a four-wheel-steering passenger car," in *Proceedings of 1995 American Control Conference (ACC)*, pp. 2292–2296.

[25] D. Xianglei, Z. Shuguang, H. Lvchang and H. Rong, "The Neural Network Direct Inverse Control of Four-wheel Steering System," in *2011 3rd International Conference on Measuring Technology and Mechatronics Automation*, pp. 865–869.

[26] D. Triantafyllidou and A. Tefas, "Face detection based on deep convolutional neural networks exploiting incremental facial part learning," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 3560–3565.

[27] Y. W. Wong, K. P. Seng and L. Ang, "Radial Basis Function Neural Network With Incremental Learning for Face Recognition," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 41, no. 4, pp. 940–949, 2011.

[28] D. Masip, À. Lapedriza and J. Vitria, "Boosted online learning for face recognition," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 39, no. 2, pp. 530–538, 2009.

[29] M. Artač, M. Jogan and A. Leonardis, "Incremental PCA for on-line visual learning and recognition," in *2002 Proceedings of the IEEE 16th International Conference on Pattern Recognition*, pp. 781–784.

[30] V. Lomonaco and D. Maltoni, "Comparing incremental learning strategies for convolutional neural networks," in *Artificial Neural Networks in Pattern Recognition 7th IAPR TC3 Workshop*, pp. 175–184, 2016.

[31] D. Lee, "Incremental robot skill learning by human motion retargetting and physical human guidance," in *2015 12th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pp. 585–586.

[32] M. Saveriano, S. An and D. Lee, "Incremental kinesthetic teaching of end-effector and null-space motion primitives," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3570–3575.

[33] S. Bose and M. Huber, "Incremental learning of neural network classifiers using reinforcement learning," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 2097–2103.

[34] Z. Wang, C. Chen, H. Li, D. Dong and T. Tarn, "A novel incremental learning scheme for reinforcement learning in dynamic environments," in *2016 12th World Congress on Intelligent Control and Automation (WCICA)*, pp. 2426–2431.

[35] R. Elwell and R. Polikar, "Incremental Learning of Concept Drift in Nonstationary Environments," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1517–1531, 2011.

[36] S. Yang and X. Yao, "Population-Based Incremental Learning With Associative Memory for Dynamic Environments," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 5, pp. 542–561, 2008.

[37] A. Gijsberts and G. Metta, "Incremental learning of robot dynamics using random features," in *2011 IEEE International Conference on Robotics and Automation*, pp. 951–956.

[38] A. Moran Cardenas, "Control of nonlinear dynamic systems using neural networks with incremental learning," in *2018 4th International Conference on Control, Automation and Robotics (ICCAR)*, pp. 182–189.

[39] X. Geng and K. Smith-Miles, *Incremental Learning*, pp. 731–735. Springer US, 2009.

[40] I. Bronstein, K. Semendjajew, G. Grosche, V. Ziegler and D. Ziegler, *Taschenbuch der Mathematik*, pp. 604–605. Springer, 2013.

[41] A. Moran Cardenas and M. Nagai, "Autonomous Driving of Truck-Trailer Mobile Robots with Linear-Fuzzy Control for Trajectory Following," in *2020 IEEE International Conference on Fuzzy Systems*, pp. 1–8.

ET LUX IN TENEBRIS LUCET
MCMXVII

List of Figures

ET LUX IN TENEBRIS LUCET
MCMXVII

List of abbreviations and symbols

Abbreviations

4WS	four wheel steering
BPTT	Back propagation through time
DBP	Dynamic back propagation
DOF	Degrees of freedom
IDL	Incremental deep learning
IL	Incremental Learning
MPC	Model predictive control
NN	Neural network

Symbols

Δt	Timestep
δ_F, δ_R, δ	Steering angle in rad
ϵ	Distance to target trajectory
η	Learning rate
Ψ	Orientation of the robot
d	Distance to $x = 0$
$f()$	Activation function
J	Cost function
L	Length of robot
m	Input of hidden Neuron
n	Output of hidden neuron
R, Q	Matrices of cost function
u	Vector of model inputs
v	Velocity of robot in m/s
v_{lij}, w_{lij}	Weight after layer l, which connects the i-th with the j-th neuron
x	Vector of model states
x_{pos}, y_{pos}	Coordinates of position of the robot

ET LUX IN TENEBRIS LUCET
MCMXVII

Statement

I hereby confirm that I have written the submitted thesis by myself, without using any sources other than those indicated. Appropriate credit has been given where reference has been made to the work of others. This thesis is part of the integrated international double degree program with the Technical University of Ilmenau and the Pontificia Universidad Católica del Perú and will therefore be submitted to both universities.

Ilmenau, February 25th, 2021 Isabella Glöde