PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

TECHNISCHE UNIVERSITÄT ILMENAU

Pontificia Universidad Católica del Perú

Escuela de Posgrado

Tesis de Maestría

Autonomous Obstacle Avoidance and Positioning Control of Mobile Robots Using Fuzzy Neural Networks

Para obtener el grado de:
Magíster en Ingeniería de Control y Automatización

Presentado por: Anna-Maria Stephanie Grebner

Tutor Responsable (TU Ilmenau): Prof. Dr.-Ing. Johann Reger

Professor Responsable (TU Ilmenau): Prof. Dr. Kai Wulff

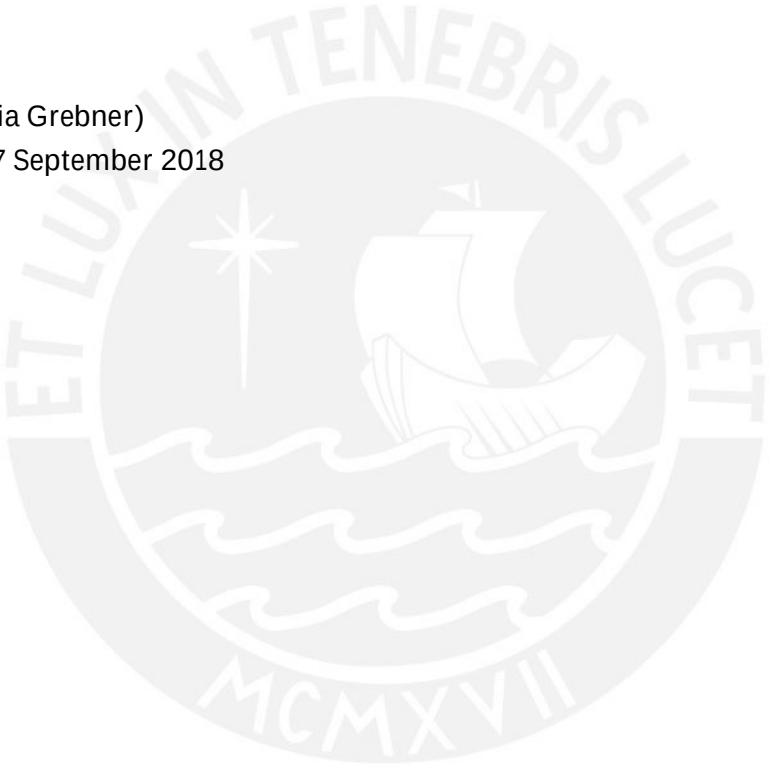
Professor Responsable (PUCP): Prof. Dr. Antonio Manuel Moran Cárdenas

Fecha y Lugar: Octubre, 2018

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties.
I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Anna-Maria Grebner)
Ilmenau, 17 September 2018



Abstract

Navigation and obstacle avoidance are important tasks in the research field of autonomous mobile robots. The challenge tackled in this work is the navigation of a 4-wheeled car-type robot to a desired parking position while avoiding obstacles on the way. The taken approach to solve this problem is based on neural fuzzy techniques.
Earlier works resulted in a controller to navigate the robot in a clear environment. It is extended by considering additional parameters in the training process. The learning method used in this training is dynamic backpropagation.
For the obstacle avoidance problem an additional neuro-fuzzy controller is set up and trained. It influences the results from the navigation controller to avoid collisions with objects blocking the path. The controller is trained with dynamic backpropagation and a reinforcement learning algorithm called deep deterministic policy gradient.

Kurzfassung

Navigation und Hindernisvermeidung sind wichtige Aufgaben im Forschungsbereich autonomer mobiler Robotik. In dieser Arbeit wird die kollisionsfreie Navigation eines Roboters mit 4 Rädern in eine Zielparkposition behandelt. Der gewählte Ansatz um das Problem zu lösen basiert auf Techniken aus dem Neuro-Fuzzy Bereich.
In früheren Arbeiten entstand bereits ein Regler mit welchem der Roboter in einer hindernisfreien Umgebung navigiert werden konnte. Dieser wurde erweitert indem zusätzliche Parameter in den Trainingsprozess mit einbezogen wurden. Die angewandte Lernmethode ist Dynamic Backpropagation. Für die Hindernisvermeidung wurde ein zusätzlicher Neuro-Fuzzy-Regler eingerichtet und trainiert. Dieser beeinflusst die Ergebnisse des Navigationsregler um Kollisionen mit Objekten zu vermeiden. Für das Training des Reglers kommen Dynamic Backpropgation und ein reinforcement learning-Algorithmus, genannt 'Deep Deterministic Policy Gradient Learning', zum Einsatz.
Das entwickelte Navigationssystem konnte in verschiedenen simulierten Szenarien getestet werden. Der Roboter war in der Lage ohne Probleme beliebige Parkpositionen anzufahren und verschieden geformte Linien zu verfolgen.

Contents

Contents

Chapter 1

Introduction

Robots are growing more and more important in our modern world. Whether to make our life easier or take over dangerous tasks, their presence in our daily life has been increasing in the recent years. Especially autonomous mobile robots can already be found in many everyday applications. Museums and stores employ them to guide visitors and floor-cleaning or lawn-mowing robots made their way in many homes. In order to move around in the environment without problems, good navigation and obstacle avoidance systems areessential formobilerobots. They guarantee that therobotreaches desired positions in time and does not collide with structures and objects on the way.

In this thesis a control system for a navigation and obstacle avoidance problem is developed, implemented and tested. The task consists of navigating an autonomous 4-wheeled cartype robot in a desired parking position, while avoiding collision with obstacles on the way. The techniques used to realize the controller belong in a category called fuzzy neural networks. They are hybrid systems consisting of elements from fuzzy inference systems and artificial neural networks. The strength of fuzzy systems lies in the possibility to create a system based on expert knowledge. Neural networks are known for their ability to learn and adapt to new situations. Neuro-fuzzy networks combine the advantages of both approaches by considering a fuzzy inference system as a neural network and apply training algorithms to improve their performace. The training methods used in this thesis are dynamic backpropagation and deterministic policy gradient.

This work is structured as follows: In chapter 2 the problem to be solved in this thesis is layed out in detail. Chapter 3 contains the theoratical foundations needed to understand this work. In chapter 4 a complete description on how the the controllers and learning algorithms were implemented is given. Chapter 5 presentes the results

reached with the controllers in different test scenarios, followed by the conclusion in chapter 6.

Chapter 2

Problem Description

The task for this thesis is to develop a neuro fuzzy controller capable of navigating a 4-wheeled robot in the x-y-plane to a goal position. The problem is designed to be parking situation in which the robot has to move backwards in its designated spot. In the first part, the path to the goal position is clear and the robot should simply be able to reach it. This problem is shown in figure 2.1. The robot should be able to get from every starting position between -50 and 50 on the x-axis and every starting orientation to the target position of $x^* = 0$ with an orientation of $\varphi^* = \frac{\pi}{2}$. This kind of problem

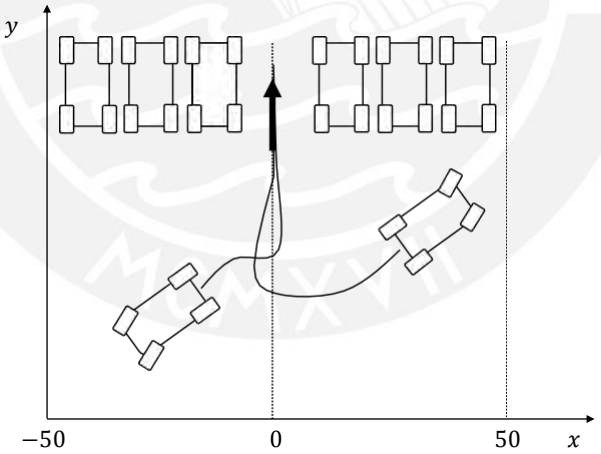


Figure 2.1 – Parking problem to be solved

was once before tackled in [1] and a neuro-fuzzy controller was developed. Dynamic backpropagation was used in training, to adapt the parameters in the consequence part of the fuzzy controller. It lacked, however, the training of the premise parameters, which is to be added in the framework of this thesis.

In the second part of the problem the robots way to the goal position is blocked by an obstacle, like shown in figure 2.2. The navigation controller is to be extended by the functionality to avoid obstacles. The solution should again be found by applying neuro-fuzzy techniques. A structure for the obstacle avoidance controller has to be set up and trained to fulfill the task.

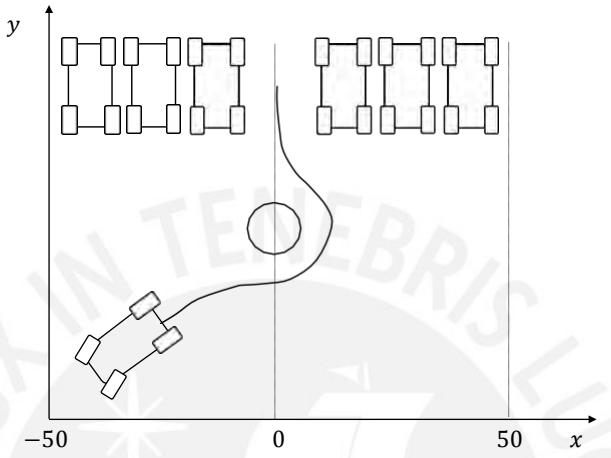


Figure 2.2 – Parking problem with obstacle avoidance

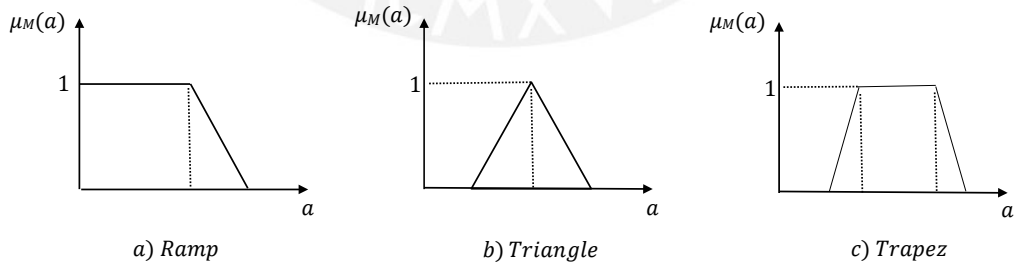
Chapter 3

Foundations

3.1 Fuzzy Inference Systems

3.1.1 Fuzzy Sets and Fuzzy Logic

Fuzzy systems, and therefore fuzzy controllers, are based on the concepts of fuzzy sets and fuzzy logic, which were first established by Zadeh in 1965 [2]. The idea is to expand the classic set theory by an uncertainty: Whereas in the original theory it is only possibile for an object a to belong to a set M or not, in fuzzy theory it can belong partially to a fuzzy set. The membership of an object to a specific set is described with the membership function $\mu_M(a)$. In conventional sets the membership function can only take the so-called 'crisp' values 0 (if $a \in M$) or 1 (if $a \notin M$). The membership grade to a fuzzy set though, can be any value between 0 and 1, allowing it to be partial or 'fuzzy'. Common forms of membership functions in fuzzy-systems are shown in figure 3.1: the ramp (a), the triangle (b) and the trapez(c).



$a)\ Ramp$ $b)\ Triangle$ $c)\ Trapez$

Figure 3.1 – Common forms of fuzzy membership functions

Fuzzy sets allow to characterize states without giving a strict classification. The human language contains a lot of options to describe things in a vague way, like e.g. 'a

little bit', 'a lot more' or 'almost'. That is why fuzzy sets are a perfect instrument for transforming human language statements into a structure that can be treated formally exact. An example is given in figure 3.2, which shows a fuzzy description of a distance. The fuzzy variable is the distance d, which can be charaterized by the fuzzy sets 'ZERO', 'NEAR' and 'FAR'. A distance of 0 gets full membership in the set 'ZERO'. A distance

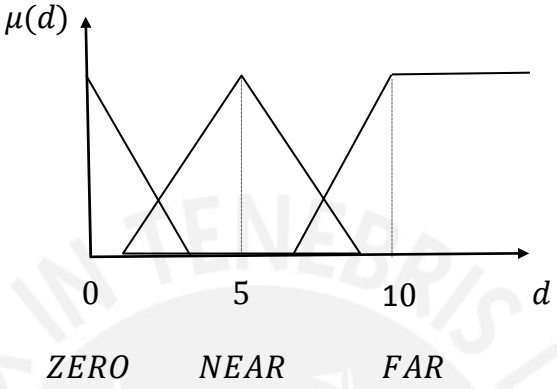


Figure 3.2 – Fuzzy description of a distance

of 5 is considered as 'NEAR' and every distance of 10 or greater is 'FAR'. The fuzzy sets allow to describe also distances in between. A distance of 3 for example would have partial membership grades in the sets 'ZERO' and 'NEAR'. It could be described as 'close to zero', 'almost zero' or 'very near'. As fuzzy sets in practise are often connected with a linguistical interpretation like this, they're also called 'linguistic variables'.

In order to work with fuzzy variables it is necessary to establish connecting operators. Crisp variables can be linked with the boolean operators from standard logic. The definition of these were extended for fuzzy variables. Different applications resulted in different forms of this adaption. The most common ones are explained below.

- AND operation ($\wedge$)
  The first definition of a fuzzy AND-operator was proposed in [2] and uses a minimum operation

$$\mu_{A\wedge B} = min(\mu_A, \mu_B) \tag{3.1}$$

  As for optimization processes differentiability is often a requirement another definition can be used

$$\mu_{A\wedge B} = \mu_A \cdot \mu_B \tag{3.2}$$

- OR operation($\vee$)
  The two most commonly used definitions for the OR-operator are the maximum operation

$$\mu_{A\vee B} = max(\mu_A, \mu_B) \tag{3.3}$$

  and the algebraic sum

$$\mu_{A\vee B} = \mu_A + \mu_B - \mu_A \cdot \mu_B \tag{3.4}$$

- NOT operation
  The NOT-operation of a membership grade is determined by the difference between 1 and the degree

$$\mu_{\overline{A}} = 1 - \mu_A \tag{3.5}$$

The tables 3.1 and 3.2 show that these definitions are consistant with the results from boolean operators and how they work on a fuzzy membership example

μ_A	μ_B	$min(\mu_A, \mu_B)$	$\mu_A \cdot \mu_B$
1	1	1	1
1	0	0	0
0	1	0	0
0	0	0	0
0.2	0.8	0.2	0.16

Table 3.1 – Results of AND-operators

μ_A	μ_B	$max(\mu_A, \mu_B)$	$\mu_A + \mu_B - \mu_A \cdot \mu_B$
1	1	1	1
1	0	1	1
0	1	1	1
0	0	0	0
0.2	0.8	0.8	0.84

Table 3.2 – Results of OR-operators

3.1.2 Fuzzy Inference Systems

Fuzzy inference systems (FIS) are systems which describe functions by mapping inputs to outputs with the help of fuzzy sets and fuzzy logic. The following information about them are taken from [3] and [4]. To explain the functionality of a FIS and example with 2 inputs and 1 output, as shown in figure 3.3,is considered.

The 3 mayor steps to determine the output are called fuzzification, inference and defuzzification. The first and the latter are ports to the surrounding environment. They realize the transformation between the real, crisp values outside and the fuzzy values

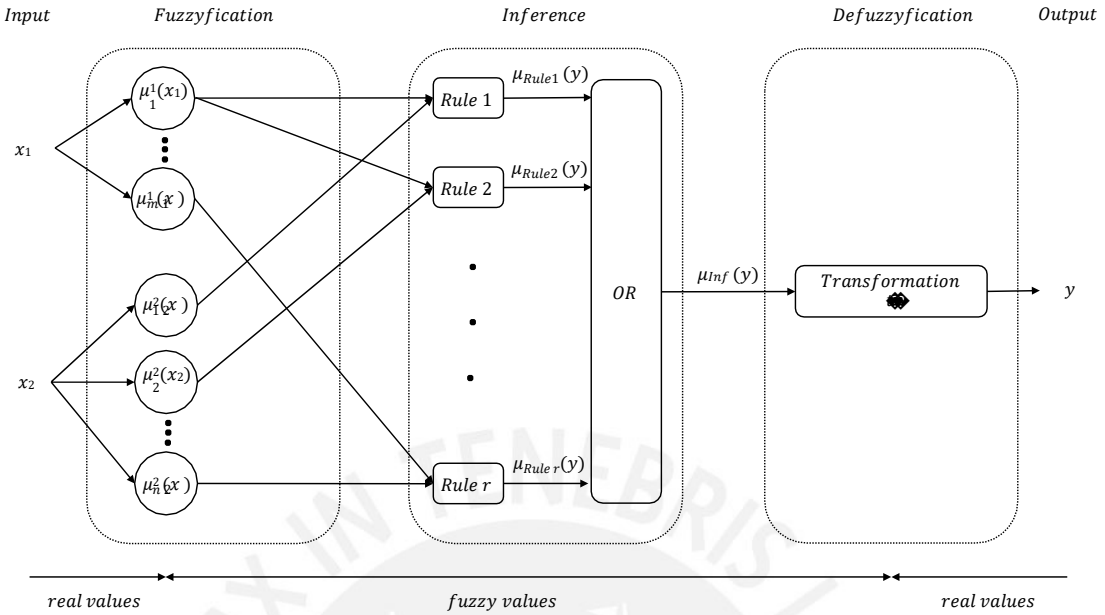


Figure 3.3 – Structure of a fuzzy inference system

inside the FIS. The inference section contains the most important part of the fuzzy system: the rule base. It determines the relation between the input an the output and therefore the behaviour of the system. The functionality of the individual phases is explained further in the following section.

Fuzziftcation

In the fuzzification step the real input values are transformed to fuzzy values. Every input variable gets several fuzzy sets, which coverthe complete inputspace. The quantity of sets and the shape of their membership functions are design parameters and should describe the input adequatly. The membership functions for one input will then look similar to the ones in figure 3.2. The fuzzy system in the structure diagram above has 2 inputs, which are characterized by respectivly m and n membership functions. For the first input x_1 they are called $\mu^1_1, \mu^1_2 \ldots, \mu^1_m$ and for the second one x_2 they are $\mu^2_1, \mu^2_2 \ldots \mu^2_n$.
The result of the fuzzification are the membership grades of all inputs to each of their respective fuzzy sets.

Inference

The heart of every FIS is the rule base. A complete rule base has a rule for every possible combination of membership grades from the fuzzification. They determine the

output of the system for a specific input vector by employing r rules in an implication (IF-THEN) structure.

IF premise i THEN consequence i

The premises (P_i) are logical combinations of the results from the fuzzification step. In the consequence (C_i) of every rule i is a membership function $\mu_i(y)$ for the output. The first rule for the system in figure 3.3 can therefore have a structure like this

$$\text{IF } \mu^1_1(x_1) \text{ AND } \mu^2_1(x_2) \text{ THEN } \mu_1(y)$$

The first step for determining the result of a rule is called aggregation and consists of calculating the premises with the fuzzy logic operators. The premises are fulfilled to different degrees, depending on the actual input vectors. Every consequence should be active at the same level as their respective premise. The logical expression, which has to be fulfilled for this demand is

$$(P_i \wedge (P_i \rightarrow C_i)) = (P_i \wedge C_i)$$

The transformation to the easier AND-operation can be done in classical logic only. It is, however, also used in fuzzy logic as an easier approximation. Depending on which of the AND-operators discussed in the previous section is used, the conclusion membership function is either cut ($min(\mu_{P_i}, \mu_{C_i})$) or scaled ($\mu_{P_i} \cdot \mu_{C_i}$) to the level of the premise (see figure 3.4). The calculation of the consequences is also called activation. In the

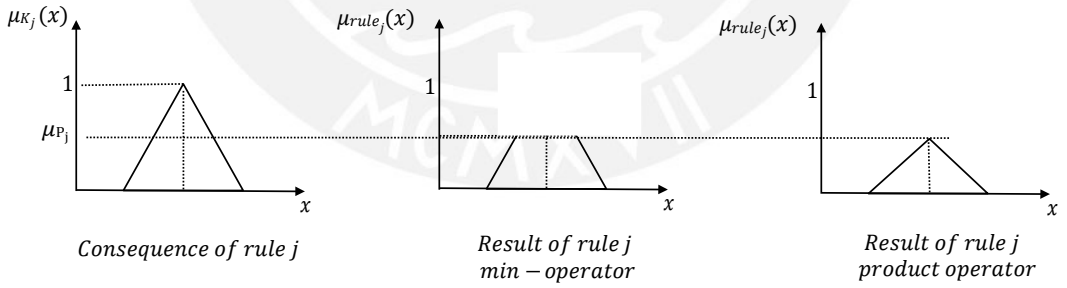


Figure 3.4 – Activation in a FIS with min- and product operator

last part of inference all the consequences have to be accumulated to one fuzzy output. This is done by employing an OR-operator over all the results from the rules.

$$\mu_{Inf}(y) = \mu_{C_1} \vee \mu_{C_2} \vee \cdot \cdot \cdot \vee \mu_{C_r}$$

The result is one membership function for the output space, defining a fuzzy value of $\hat{y}$

Defuzziftcation

This membership function needs to be transformed into a real value for the output $\hat{y}$. The result should consider all the rules consequences in their respective strenght. One method to calculate a real output is to determine the center of gravity of the accumulated membership function. This can be done with the following formular:

$$\hat{y} = \frac{\int_{-\infty}^{\infty} \mu_{Inf}(y) \cdot y \, dy}{\int_{-\infty}^{\infty} \mu_{Inf}(y) \, dy} \tag{3.6}$$

As this approach needs a lot of computing time, another way of determing the output can be found in the maximum method. It just takes the output value corresponding to the maximum value of $\mu_{Inf}(y)$. This method is easy but has the disadvantage that only the rule producing the maximum is considered for the out come. Also there is not a unique solution if the maximum is a plateau. In this case it is possible to define the mean of the maximum as the output.

A special form of fuzzy inference systems are Takagi-Sugeno fuzzy systems, in which the concequences as not described by membership functions but by a linear combination of the real input values:

$$\text{IF } P_i \text{ THEN } \hat{y} = y_{i0} + y_{i1}x_1 + y_{i2}x_2 + \cdots + y_{il}x_l$$

With these type of systems an accurate approximation of functions can be reached. However, this type of concequence also reduces the comprehensibility of the system. In Takagi-Sugeno systems the defuzzification simplified so much, it can be concluded with the inference to the following

$$\hat{y} = \frac{\sum_{i=1}^{*} \mu_{P_i} \cdot (y_{i0} + y_{i1}x_1 + y_{i2}x_2 + \cdots + y_{il}x_l) \, dy}{\sum_{i=1}^{*} \mu_{P_i}} \tag{3.7}$$

Takagi-Sugeno systems, in which $y_{i1} = y_{i2} = \cdots = y_{il}$ are called 'zero order Takagi-Sugeno systems'. They have crisp values in their consequences, which are expressed as singleton functions.

$$\text{IF } P_i \text{ THEN } \hat{y} = y_i$$

Fuzzy inference systems can be employed for a variety of applications like modeling,

controlling or classification. The advantage of fuzzy systems is the easy to understand structure. The possibility to describe input-output-relationships intuitively as simple rules makes FIS the perfect instruments to induce human knwoledge in a process. By checking every rule it is also easy to check if the system is working in the correct and desired way for the whole input space. The disadvantages are, that an expert is needed to define the rules and human knowledge alone often ist not at the optimum.

3.2 Neural Network

3.2.1 Structure of Neural Networks

Artificial neural networks are a method to process data, which is inspired by the way human brains work with information. The following section serves to describe the fundamental structure and functionality of artificial neural networks, as well as the process of training them. It is based on the information in [4].
Biological neural networks consist of connected nerve cells, called neurons, which handle given stimulations and pass them on to connected neural cells. In the same way artifical networks are build up of connected smallest units, called neurons. The structure of a neuron can be seen in figure 3.5. It can have multiple inputs (x_1, x_2, x_3 in the figure),

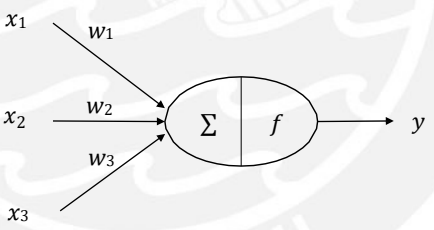


Figure 3.5 – Neuron, smallest unit of a Neural Network

which either come from other neurons or the outside of the network. Every input is associated with a weight (w_1, w_2, w_3). The neuron consists of two functions. The first one combines the inputs by multiplaying each input with its respective weight and adding the results up

$$x_c = \sum_{i=1}^{number\ of\ inputs} x_i w_i$$

The other function is called activation f and uses the combined input to generate the output. Activation funtions are often nonlinear to include the possibility to handle nonlinear relations with the network. For this purpose a sigmoid function (also called

fermi function) is frequently used:

$$f(x_c) = \frac{1}{1 + e^{-ax_c}}$$

An artificial neural network forms when multiple of these neurons are connected and interchange their results. The easiest form to do this are feedforward multilayer networks, which can be seen in figure

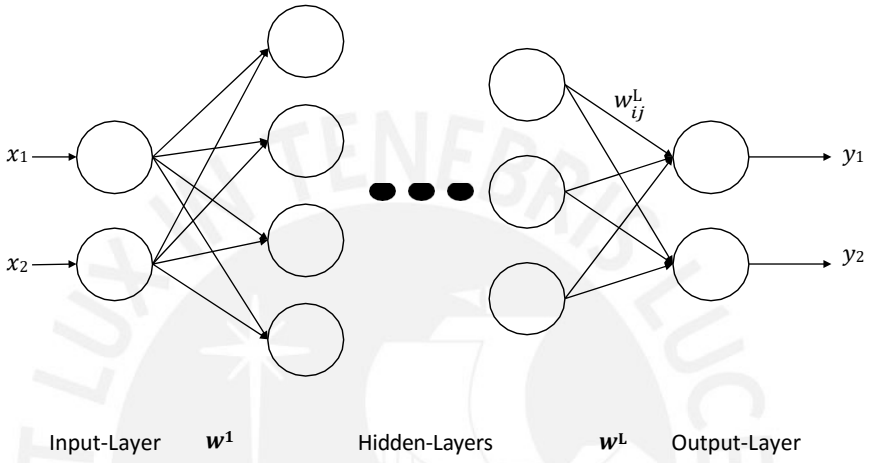


Figure 3.6 – Feedforward Network

There are three different types of layer in a multilayer network. The input layer takes in the data given to the network and provides them for the following layers. Therefore the number of neurons in this layer equals the number of inputs to the network. After the input layer follow any number of hidden layers, which absolve the computation of data. The final layer is called output layer and consists of as many neurons as the system has outputs to give them back at the environment. To compute data with the neural network, the data is given from layer to layer, which is called a forward pass. The output of every neuron in one layer is calculated before the processing in the next layer begins. Due to their ability to learn neural networks can be used in a variety of application, for example in modelling, speech/writing recognition or controllers. Their disadavantage is, that it is difficult to keep track on the learning process and ensure that the learned parameters are, in fact, correct ones.

3.2.2 Standard Backpropagation

An important step in the implementation of neural networks is the training. This describes the process of adapting the parameters in the network, so it will behave in the

desired way. One of the most basic ways to do this is error-backpropagation. The goal of this method is to adapt the weights at the connections between the neurons. This training method belongs to the techniques of supervised learning. These algorithms require that the desired output of the network is known so it can be compared to the real output. For the training process N pairs of input values (x) and desired output values, called teacher vector (t), are combined to a training set. The input values are presented to the network and the output is calculated via forward pass. The output can be written as a function of the input and the current network parameters (θ).

$$y = g(x, \theta) \tag{3.8}$$

The goal of training is to minimize the error of the whole training set

$$E(\theta) = \frac{1}{2} \sum_{i=1}^{N} (t_i - g(x_i, \theta))^2 \tag{3.9}$$

bybackpropagating it through thenetworkinanattempt toadjusttheweightresponsible for it. Therefore a gradient method is used: The weights are adapted in their proportion in direction of decline in the error

$$\theta_{k+1} = \theta_k + \Delta\theta \tag{3.10}$$

$$\Delta\theta = -\eta \frac{\delta E}{\delta \theta} \tag{3.11}$$

In the second equation, η is the learning rate, which determinates the size of steps made in the gradient direction. The derivative of E is computated via chain rule. For adapting the weight w_{ij} connecting the j-th neuron in the L-th layer with the i-th neuron in the $(L + 1)$-th layer it becomes

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \frac{\partial f(x_j)}{\partial x_j} \frac{\partial x_j}{\partial w_{ij}} \tag{3.12}$$

$$= \delta \frac{dx_j}{dw_{ij}} \tag{3.13}$$

$$= \delta_j y_j \tag{3.14}$$

where x_{jc} the combined input and $y_j = f(x_{jc})$ is the output of the j-th neuron in the L-th layer. The δ_j in the last equation can be computed in a recursion with values from

the $(L+1)$-th layer.

$$\delta_j = \frac{\partial f(x_j)}{\partial x_j} \sum_{k \leftarrow j} \omega_{jk} \delta_k^{(L+1)} \tag{3.15}$$

All these values will be available, as for backpropagation one starts at the output and work the way torward the input layer until all weights are adapted. With the δ-Notation, it is also often called Delta-Learning-Rule.

3.2.3 Dynamic Backpropagation

In chapter 3.2.2 the basic learning process of neural networks via backpropagation was explained. An extended method is dynamic backpropagation, which is explained in [5] andused for the training ofrecurrent neuralnetworks. This technique takes intoaccount that in this case information get fed back to the neural networks and works with this information from the past. When neural networks are used as controllers, they are also in a closed circle, which is why this form of training can be applied here. To explain this algorithm the whole process and interactions between controller and system are unfolded over time in figure 3.7. When unfolded over time, the control loop becomes a

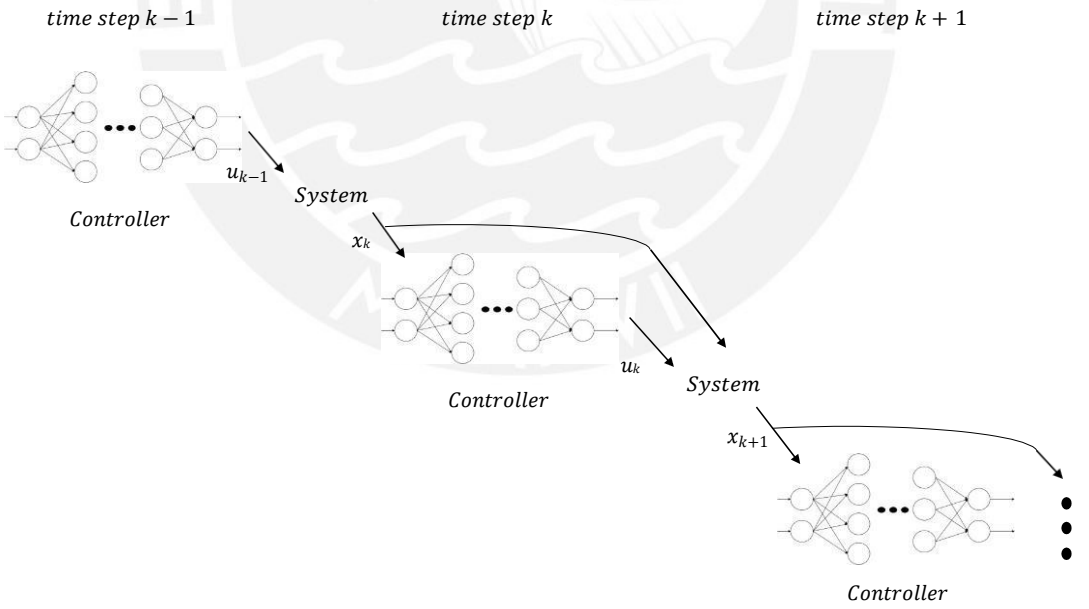


Figure 3.7 – Unfolded Network

series of repetition between the controller and the system. This means, that their errors

depend not only on the current parameters but also on past ones. Therefore for their value adaption, full derivatives until the starting point have to be calculated. This can be done in a recursive algorithm, which is explained below. The goal is to minimize the difference between the current and the desired position in every time step. This results in a cost function of

$$J = \frac{1}{2}(x_1 - x_1^*)^2 + \frac{1}{2}(x_2 - x_2^*)^2 + \ldots + \frac{1}{2}(x_N - x^*_N)^2 \tag{3.16}$$

As before in standard backpropagation, the weights are adapted with

$$w_{ij,(k+1)} = w_{ij,k} - \Delta w_{ij,k} = w_{ij,k} - \eta \frac{\overline{\partial J}}{\partial w_{ij}} \tag{3.17}$$

The overline in the last derivative signalizes, that the derivative is not only calculated to the actual influence, but is the total partial derivate considering all w_{ij} in the unfolded system. With equation 3.16 this total derivative can be split up to

$$\frac{\overline{\partial J}}{\partial w_{ij}} = \sum_{k=1}^{N} (x_k - x^*_k) \frac{\overline{\partial x_k}}{\partial w_{ij}} \tag{3.18}$$

The total derivative of x_k is

$$\frac{\overline{\partial x_k}}{\partial w_{ij}} = \frac{\partial x_k}{\partial w_{ij}} + \frac{\partial x_k}{\partial x_{k-1}} \frac{\overline{\partial x_{k-1}}}{\partial w_{ij}} \tag{3.19}$$

$$= \frac{\partial x_k}{\partial w_{ij}} + \sum \left(\frac{\partial x_k}{\partial x_{k-1}} + \frac{\partial x_k}{\partial u_{k-1}} \frac{\partial u_{k-1}}{\partial x_{k-1}} \right) \frac{\overline{\partial x_{k-1}}}{\partial w_{ij}} \tag{3.20}$$

Leading to the recursive formular for computing the total derivative of the state to the weight in dynamic backpropagation for neural controllers:

$$\frac{\overline{\partial x_{k+1}}}{\partial w_{ij}} = \frac{\partial x_{k+1}}{\partial u_k} \frac{\partial u_k}{\partial w_{ij}} + \left(\frac{\partial x_{k+1}}{\partial x_k} + \frac{\partial x_{k+1}}{\partial u_k} \frac{\partial u_k}{\partial x_k} \right) \sum \frac{\overline{\partial x_k}}{\partial w_{ij}} \tag{3.21}$$

During training $\frac{\partial x_{k+1}}{\partial u_k}$ and $\frac{\partial x_{k+1}}{\partial x_k}$ are determined with the system model equations. The other derivatives - $\frac{\partial u_k}{\partial w_{ij}}$ and $\frac{\partial u_k}{\partial x_k}$ - are calculated by backpropagation through the controlling neural network.

3.2.4 Reinforcement Learning with an Actor-Critic-Algorithm

In the last two chapters techniques of backpropagation were explained as examplse of a supervised learning process. However, one may not always know the best value for the output of a network for comparison. For example in the case of controllers the desired

state may be known, but not the value of the actuating variable to reach. Unsupervised learning techniques attempt to solve this problem. A group of these methods is called reinforcement learning. These algorithms let the network operate and observate how it influences the environment. If the result of the taken actions are satisfactional the positive feedback is given, otherwise a negative one. Information about reinforcement learning in general can be found in [6]

The reinforcement algorithm which is presented below is called 'Deep Deterministic Policy Gradient' and was developed in [7]. It is based on [8] from 2014, in which the deterministic policy gradient was first presented. The algorithm was chosen because it can be employed for continuous state and action spaces, which are found in navigation problems. The algorithm has an Actor-Critic structure (see figure 3.8). The 'actor' is

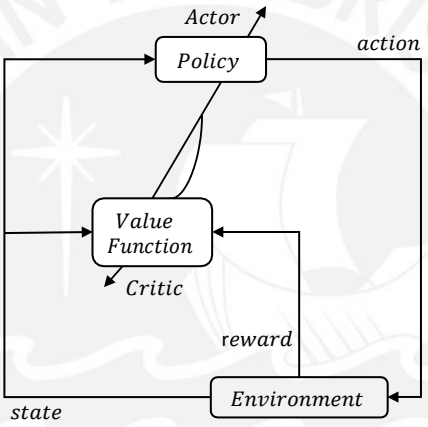


Figure 3.8 – Structure of an Actor-Critic network

the network which shall be trained. Based on a policy $\mu_a(s)$, it decides which action a is taken in a specific state s. A deterministic policy can simply be seen as the actor network with its current structure and parameters, computing an action as an output.

$$a_k = \mu_a(s_k) \tag{3.22}$$

The action a_k effects the environment and an immediate feedback is given in form of a reward r_k. The reward can be positive or negative, depending on weather a_k helped the system to reach a defined ojective. An ideal policy μ_a^* would chose actions which lead to the highest accumulated reward R in a long term horizon. To adapt the current policy in the direction of the ideal policy another network is employed, the so-called 'critic'. It evaluates the taken action in every time step by estimating the action-value function $Q(s, a)$, which represents the expected long-term reward if action a is taken at state s.

The ascending gradient of this function leads in the direction of the ideal policy and is therefore used for adapting the current policy. The problem is, that the action-value function is not known in the beginning and the critic network has to learn it. The ideal action-value function returns the highest accumulated reward to be expected under the current policy:

$$Q^*(s_k, a_k) = max_{\mu_a} E[R|s_k, a_k] \tag{3.23}$$

This optimal function fullfills the so-called Bellmann equation

$$Q^*(s_k, a_k) = E[r_k + \gamma Q^*(s_{k+1}, a_{k+1})] \tag{3.24}$$

with γ being a discount factor between 0 and 1. The ideal value can be calculated with this equation and the difference to the current output can be backpropagated through the critic network. As the action value $Q^*(s_{k+1}, ak + 1)$ and therefore the action a_{k+1} of the next time step is needed, two additional networks are implemented. These are called target networks and estimate the future values. With the parameters of the actor and critic networkbeing θ_a and θ_c the target networks parameters are designed to slowly follow them and updated via

$$\theta_a^t = \tau\theta^t{}_a + (1 - \tau)\theta^t{}_a \tag{3.25}$$

$$\theta_c^t = \tau\theta^t{}_c + (1 - \tau)\theta^t{}_c \tag{3.26}$$

with $\tau << 1$. Th outputs of the target networks can then be defined respectively as $\mu^t_a(s_{k+1}, ak + 1, \theta^t_a)$ and $Q^t(s_{k+1}, a_{k+1}, \theta^t_c)$ With these information it is possible to train the actor and critic network as described below for the time step k. The loss function at the critic network output is constructed with the mean square error

$$L(\theta_c) = E[(r_k + \gamma Q^t(s_{k+1}, a_{k+1}) - Q(s_k, a_k, \theta_c))^2] \tag{3.27}$$

The critic network is adapted with the gradient to its parameters

$$\frac{\partial L(\theta_c)}{\partial \theta_c} = E[(r_k + \gamma Q^t(s_{k+1}, a_{k+1}) - Q(s_k, a_k, \theta_c)) \frac{\partial Q(s_k, a_k, \theta_c)}{\partial \theta_c}] \tag{3.28}$$

$$\theta_c = \theta_c + \eta_c \frac{\partial L(\theta_c)}{\partial \theta_c} \tag{3.29}$$

The parameters of the critic network are updated with the gradient of the action-value function:

$$\frac{\partial Q(s_k, a_k, \theta_a)}{\partial \theta_a} = \frac{\partial Q(s_k, a_k, \theta_a)}{\partial a_k} \frac{\partial a_k}{\partial \theta_a} \tag{3.30}$$

$$\theta_a = \theta_a + \eta_a \frac{\partial Q(s_k, a_k, \theta_a)}{\partial \theta_a} \tag{3.31}$$

As the estimated policy is deterministic it does not explore on itself. This means the actor choses the same action in the same situation every time. As it is necessary to try different actions in order to find the best policy a random process is added as a noise to the taken actions to ensure sufficient exploration. In [7] an Ornstein-Uhlenback random process is named to be suitable for this kind of task. As the random values generated by this process are temporally correlated they let the actor explorate efficiently in one direction.
In practice there are difficulties with updating the networks every time step with the new information, which are rooted in the temporal dependence of the incoming data. This is why [7] proposes to let the actor sample transitions in a form like (s_k, a_k, r_k, s_{k+1}) and store them in a replay memory. For learning some samples from the replay memory are randomly chosen in a minibatch for learning.

3.3 Fuzzy Neural Networks

Fuzzy neural networks are hybrid systems of the both mentioned structures. The idea is to model a fuzzy inference system as a neural network. The parameters of the system can then be adapted by employing learning strategies. This kind of architecture was first presented in [9] as an 'Adaptive-Network-Based Fuzzy Inference System' (ANFIS). For explaining the functionality a system with 2 inputs(x_1 and x_2), 1 output($\hat{y}$) and two rules is considered. The fuzzy sets for the input are A_1 and A_2 for x_1 and B_1 and B_2 for x_2. The structure is depicted in figure 3.9

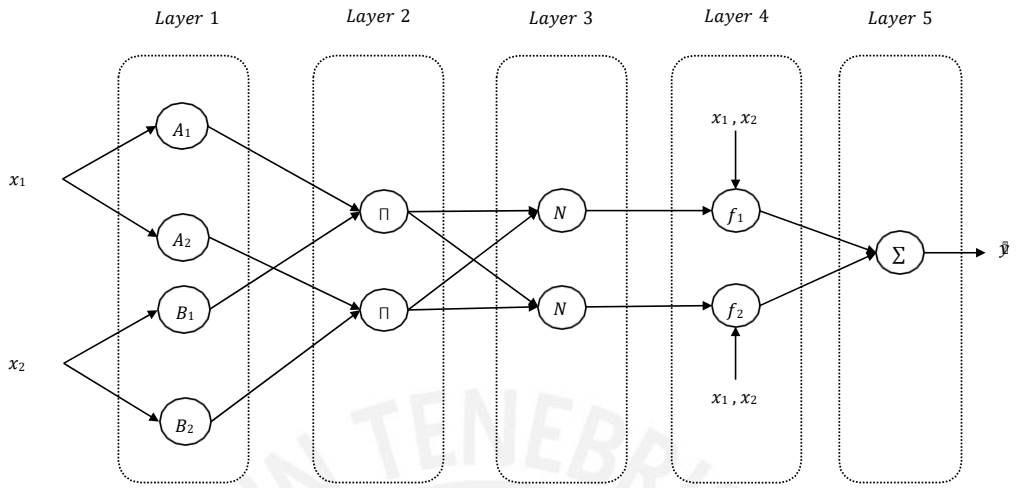


Figure 3.9 – Fuzzy-Neural-Network

The fuzzy neural network consists of 5 layers in a feed-forward structure containing different types functions.

The nodes in the first layer contain the membership functions and take over the task of fuzzification. Note that because the networkshall be trained, the membership functions need to be at least piecewise differentiable. Thus the proposed fuzzy membership functions (see chapter 3.1.1) can be employed here but are nevertheless often approximated with other, differentiable functions. For instance the triangle shape can be expressed with a gaussian function

$$\mu(x) = e^{-\frac{(x-c)^2}{a}}$$

and the ramp with a sigmoid function

$$\mu(x) = \frac{1}{1 + e^{-\frac{x-c}{a}}}$$

The variables c and a are different for everymembership function as they determine the center (c) and slope (a) of the function. From the view of fuzzy systems they define the position and size of the fuzzy sets for the input space. These parameters are called 'premise parameters' and are adaptable during training.

The second layer realizes the aggregation of the premises by using AND-operators in the nodes. In the figure above the AND is realized with a product. The result of this

layer for rule i is called 'firing strength z_i of rule i'.

$$z_i = \mu_{A_i} \cdot \mu_{B_i}$$

The third layer contains normalization nodes. They scale the firing strenghts by dividing it by the sum of all firing strengths. The outputs of this layer are referred to as 'normalized firing strengths'. This step realizes the easy defuzzification for Takagi-Sugeno fuzzy systems, described in 3.1

$$z_i^N = \frac{z_i}{z_1 + z_2}$$

The fourth layer calculates the rules cosequences and represents the inference from FIS. As ANFIS is based on Takagi-Sugeno fuzzy systems, the rules have linear combinations of the input variables as consequences.

$$\text{IF } A_1 \text{ AND } B_1 \text{ THEN } f_1 = p_1 x_1 + q_1 x_2 + w_1$$

$$\text{IF } A_2 \text{ AND } B_2 \text{ THEN } f_2 = p_2 x_1 + q_2 x_2 + w_1$$

As every rule should be active in the same way their premise is active, the consequences are multiplied with the normalized firing strengthes.

$$z_i^N f_i = z_i^N (p_i x_1 + q_i x_2 + w_i)$$

The parameters p_i, q_i and w_i are called 'consequence parameters' and can be changed by training to adapt the network.
The fifth and last layer concludes all rules consequences into one output by adding them up. The final output is therefore:

$$\hat{y} = \sum_i z_i^N f_i = \frac{\sum_i z_i f_i}{\sum_i z_i}$$

Fuzzy-Neural-Networks combine the advantages of fuzzy and artificial neural systems. The structure and functionality is easily comprehensible and can be set up without problem with parameters from expert knowledge. By having the possibility to change parameters with a training sequence the system can learn and adapt.

3.4 Neuro-Fuzzy-Control in Robot Navigation

Hybrid networks with fuzzy and neural elements have been successfully implemented in many systems for control purposes. The applications range from the control of household

devices like washing mashines [10] to process control in power plants [11]. Another branch benefiting from neuro-fuzzy-techniques is the field of autnomous mobile robots. Fuzzy neural controllers can help to tackle the challenges of navigation and obstacle avoidance, and have therefore been employed several times for these tasks.
In [12] an ANFIS-Controller was developed for the navigation of a mobile service robot with two differential wheels. It was trained to recognize and follow a line on the floor to its goal position. To realize this the desired motor speed values were predefined in a teacher vector. The fuzzy controller was then adapted in premise and consequence parameters of the rule base via supervised learning and could eventually lead the robot along the line. An approach for an obstacle avoidance controller is given in [9], where a two-wheeled robot should be able to move in a fixed area without colliding with the objects lying within. The parameters of the membership functions and the consequence part were adpated with supervised learning. The controller was tested in simulation and also experimentally. The authors in [13] focused on the adaption of the parameters in the membership functions. They predefined all parameters and set a perfect route to a goal position without colliding with an obstacle. This path was then used for adapting position and slope of the membership functions to "smooth the trajectory generated by the fuzzy logic model". The controller was employed sucessfully in a two wheeled, differential driven robot. To decrase the number of rules in one fuzzy controller the navigation algorithm in [14] is split up in a hierarchial system. Within this, different behaviours like e.g. "goto-target" or "turn-corner" are realized with respective neuro-fuzzy controllers. By doing this they could avoid having one big rule base to include all navigation orders.
Reinforcement approaches for the navigation problem were taken in [15], with using adapted versions of Q-Learning and Sarsa-Learning algorithms. Another example is [16], where a controller for line following of a two wheeled, differential driven robot could be developed and tested in simulation. A 4-wheeled non-holonomic car-type robot, like the one considered in this thesis is the subject in [17]. A neural fuzzy controller was trained with a supervised training algorithm to find its way to a goal position and avoid obstacles in its path. The controller was implemented in a real experiment environment and could successfully drive to the target point without colliding with any objects. The paper [1], which builds the foundation for this thesis, describes the developement of a neural fuzzy controller for navigating a 4-wheeled robot from differentstarting positions and angles in a goal parking position. The goal for this thesis is to develop the existing controller further and add the ability to avoid obstacles.

Chapter 4

Implementation

4.1 Model

For the development of the navigation algorithms a model of the system was needed. Figure 4.1 shows the robot considered in this work. The two back wheels are fixed and connected to the motor to provide the traction while the two front wheels are steerable. At any time the state s of the robot is defined by its position and orientation in the ground coordinate system. The position is described by its x- and y-coordinate and the orientation by the angle φ .

$$s_k = \begin{pmatrix} x_k \\ y_k \\ \varphi_k \end{pmatrix} \tag{4.1}$$

The model should describe the dynamical transition of the robots state from one time step to another based on the actual state and the steering angle δ as the control input. The kinematic equations of cartype-mobile robots are derived in [1]. With the robot moving backwards and only a very small, fixed distance r every time step the model results in

$$x_{k+1} = x_k + r\cos(\varphi_k) \tag{4.2}$$

$$y_{k+1} = y_k + r\sin(\varphi_k) \tag{4.3}$$

$$\varphi_{k+1} = \varphi_k - \frac{r}{L}\tan(\delta_k) \tag{4.4}$$

(4.5)

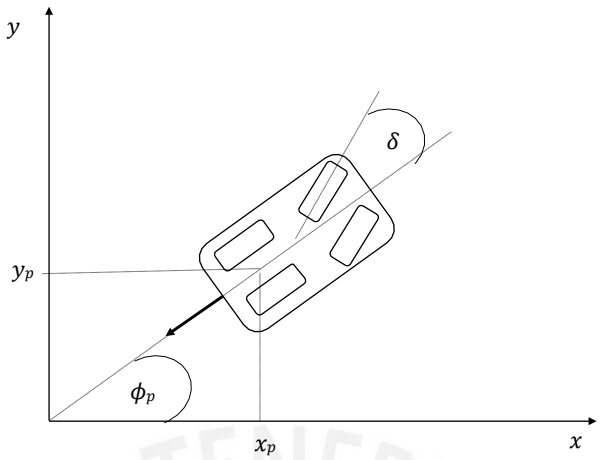


Figure 4.1 – 4-wheeled car-type robot

where L the length of the robot measured between the front and back wheels and δ_k the current steering angle input. The steering angle is restricted to $-30° < \delta < 30°$ This model applies for a slow motion of the robot at constant speed. Also it doesn't consider slipping and skidding of the wheels.

4.2 Navigation

4.2.1 Structure of the Neuro Fuzzy Controller

The strategy for solving the problem is to let the robot move to the position of $x^* = 0$ while reaching the desired orientation of $\varphi^* = \frac{\pi}{2}$. After that position is reached, the robot will just move straight in positive y-direction and the control variable will be $\delta = 0°$. It can be seen in figure 4.2 that the taken path is independent of the robots y-position. Therefore the controller does not need the y position to determine the steering angle and uses only the x-position and the angle as inputs. As long as there is enough space in y-direction the robot will be able to reach its goal with this information. From here on the position in x-direction and the orientation of the robots are concluded in the state-vector s:

$$s_k = \begin{matrix} \Sigma x_k \Sigma \\ \varphi_k \end{matrix} \tag{4.6}$$

The structure of the Neuro-Fuzzy-controller is given in figure 4.3. It has the same 5 layers as the standard neuro fuzzy system described in 3.3. The only difference is, that the controller used here is based on a zero-order Takagi-Sugeno fuzzy system. Therefore

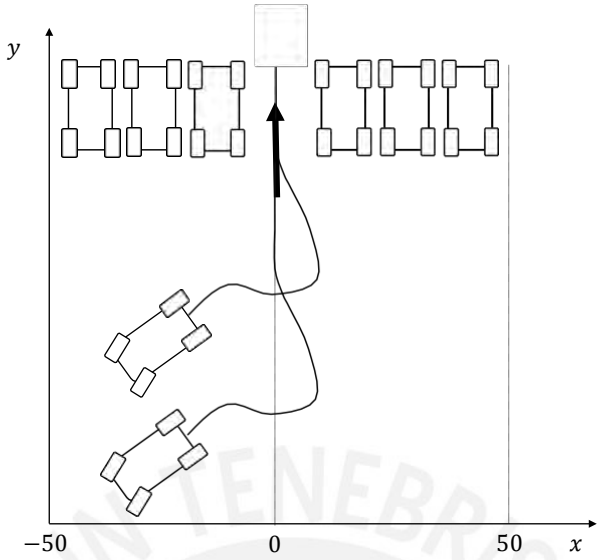


Figure 4.2 – Strategy to reach the goal position. The control input does not depend on the y-position of the robot, only on the x-position and orientation.

the consequences are not a linear combination of the inputs, but reduced to one single value w_i per rule.

Seven membership partitions were used for each of the 2 inputs. The initial membership functions used for x and φ can be seen in figure 4.4. Table 4.1 lists their center and slope values as well as the respective linguistic interpretation. The outer Membership-functions of the position are sigmoid functions, because for every x-value smaller then -50/greater than 50 the steering command should be full right/full left, the same as at $-50/50$. The outer membership functions of the orientation are gaussian, because the angle range is only 2π and the value at the left end is the same as the value at the right end.

	c_x	a_x	linguistic interpretation	c_φ	a_φ	linguistic interpretation
1	−50	4	FAR LEFT	-1.042	0.1745	DOWN RIGHT
2	−34	10	LEFT	-0.2618	0.3491	RIGHT
3	−13	6	CLOSE LEFT	0.9599	0.3491	UP RIGHT
4	0	3	ZERO	1.5708	0.0524	UP
5	13	6	CLOSE RIGHT	2.1817	0.3491	UP LEFT
6	34	10	RIGHT	3.4034	0.3491	LEFT
7	50	4	FAR RIGHT	4.1888	0.1745	DOWN LEFT

Table 4.1 – Initial center and slope parameters as well as linguistic interpretation of the membership functions for the navigation control system

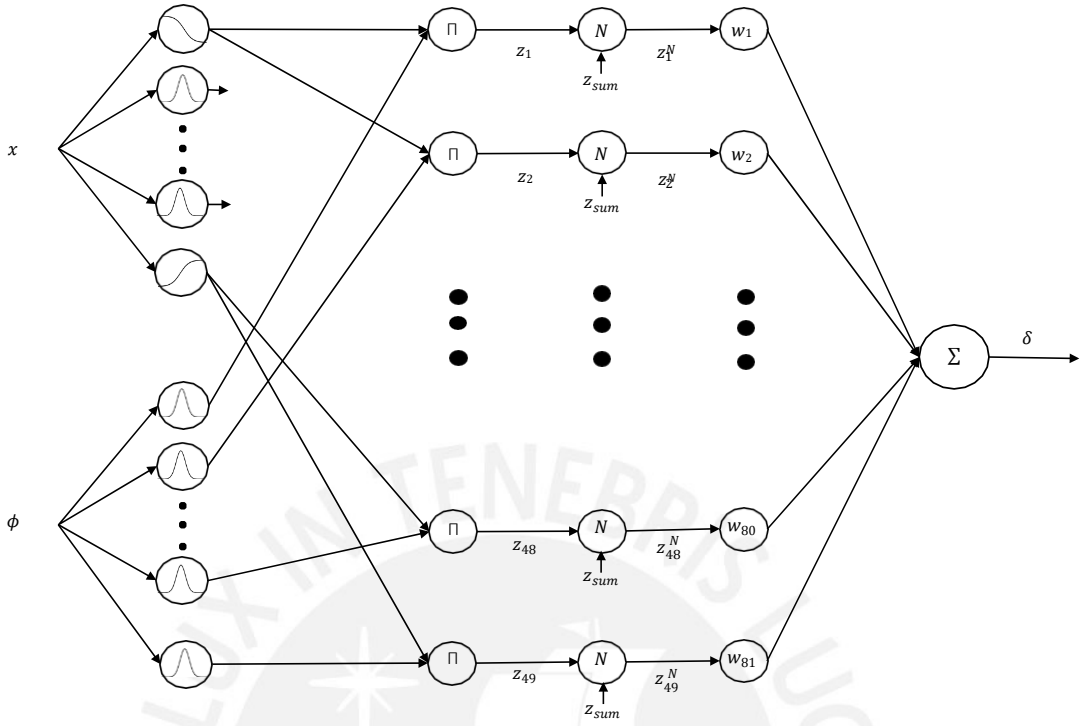


Figure 4.3 – Structure of the Neuro Fuzzy Controller

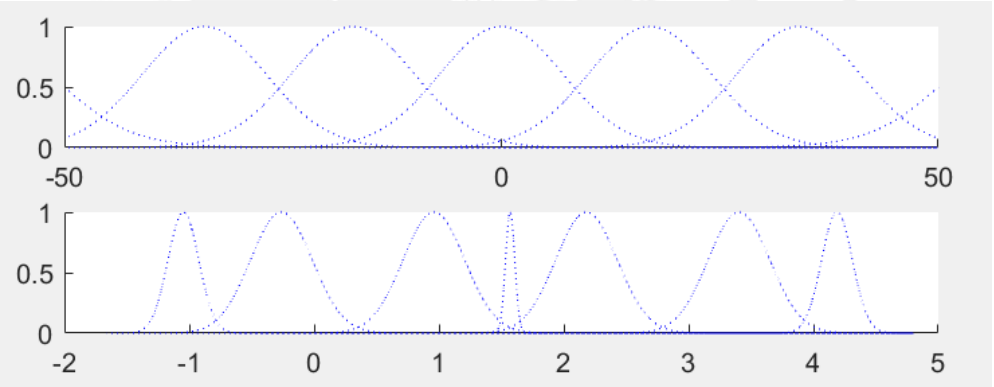


Figure 4.4 – Initial membership functions of the x-position (above) and the orientation φ (below)

The results of the membership functions are aggregated by using the product as a fuzzy AND-operator. The results are the premises $z_1 \cdots z_{49}$. These are normalized in the next layer as a part of the defuzzification.

$$z_i^N = \frac{z_i}{z_{sum}} \tag{4.7}$$

with $z_{sum} = \sum_{j=1}^{49} z_j$. Afterwards the normalized premises are multiplied with their respective weight w_i to determine the results of theirs rules. In the final step all of the results are added up to form the steering angle output. Because of the normalization step every weight w_i gets as much power on the output as their premise was active.

4.2.2 Training with Dynamic Backpropagation

The navigation controller was trained with dynamic backpropagation as described in chapter 3.2.3. The whole training process is listed in algorithm 4.1 and will be explained further here.

At first the vectors with the slopes (*a*) and centers (*c*) of the membership functions as well as of the rules consequences (*w*) are initialized. Five symmetric starting states are defined as well as the maximum of training episodes.

One training episode consist of following all starting positions for 1350 time steps or until the path leaves the field boundaries of $x_{min} = -50$ and $x_{max} = 50$. The network parameters are updated after every episode with the ratio of the sum over all calculated changes from the timesteps in this episode to the number of time steps included in this sum (k_{total}).

$$w_i = w_i - \eta \frac{\sum_{k=1}^{k_{total}} \Delta w_{i,k}}{k_{total}} \tag{4.8}$$

$$a_i = a_i - \eta_a \frac{\sum_{k=1}^{k_{total}} \Delta a_{i,k}}{k_{total}} \tag{4.9}$$

$$c_i = c_i - \eta_c \frac{\sum_{k=1}^{k_{total}} \Delta c_{i,k}}{k_{total}} \tag{4.10}$$

In each time step k the current state s_k is fed to the controller, which computes the steering angle $u_k = \delta_k$ with the available parameters. The steering angle is given to the model of the system, which calculates the next state $s_{k+1} = f_m(s_k, u_k)$ with the kinematic equations from 4.1. This position is compared with the desired position and the error e_k is determined.

$$e^k = \begin{matrix} \Sigma \\ x_{k+1} \\ \varphi_{k+1} \end{matrix} - \begin{matrix} \Sigma \\ x^* \\ \varphi^* \end{matrix}$$

With the error and the equations from the dynamic backpropagation algorithm, the gradient toadapttheparameters canbe calculated. Belowtheequations for adapting the

a rule consequence parameter w_i in the navigation controller will be derived. Differences fo the derivation of the equations for the premise parameters will be listed afterwards. As explained in chapter 3.2.2, the rate of change is the total derivative of the cost function to the parameter. For consequence i that is

$$\Delta w_i = \frac{\partial \overline{J}}{\partial w_i} \tag{4.11}$$

The cost function to minimalize consists of the error to the desired state.

$$J = \frac{1}{2}(s_{k+1} - s^*)^2 \tag{4.12}$$

With the application of chain rule the total derivative becomes

$$\frac{\partial \overline{J}}{\partial w_i} = (s_{k+1} - s^*)\frac{\partial \overline{s_{k+1}}}{\partial w_i} \tag{4.13}$$

As derived in chapter 3.2.2 this is calculated in a recursive way:

$$\frac{\partial \overline{s_{k+1}}}{\partial w_i} = \frac{\partial s_{k+1}}{\partial w_i} + \frac{\partial s_{k+1}}{\partial s_k} + \frac{\partial s_{k+1}}{\partial u_k}\frac{\partial u_k}{\partial s_k}\frac{\partial \overline{s_k}}{\partial w_i} \tag{4.14}$$

$\frac{\partial s_{k+1}}{\partial w_i}$ is split up by applying the chain rule. The first derivative is then calculated from the system model, the second from the controller structure:

$$\frac{\partial s_{k+1}}{\partial w_i} = \frac{\partial s_k}{\partial u_k}\frac{\partial u_k}{\partial w_i} = \begin{pmatrix} 0 \\ -\frac{r}{L} \end{pmatrix} z^N_i \tag{4.15}$$

The next two derivatives are constructed as well with the help of the system model

$$\frac{\partial s_{k+1}}{\partial s_k} = \begin{bmatrix} \frac{\partial x_{k+1}}{\partial x_k} & \frac{\partial x_{k+1}}{\partial \varphi_k} \\ \frac{\partial \varphi_{k+1}}{\partial x_k} & \frac{\partial \varphi_{k+1}}{\partial \varphi_k} \end{bmatrix} = \begin{pmatrix} 1 & r \cdot \cos(\varphi_k) \\ 0 & 1 \end{pmatrix} \tag{4.16}$$

$$\frac{\partial s_{k+1}}{\partial u_k} = \begin{bmatrix} \frac{\partial x_{k+1}}{\partial u_k} \\ \frac{\partial \varphi_{k+1}}{\partial u_k} \end{bmatrix} = \begin{pmatrix} 0 \\ -\frac{r}{L} \end{pmatrix} \tag{4.17}$$

For the last derivative the structure of the controller is used again.

$$\frac{\partial u_k}{\partial s_k} = \begin{bmatrix} \frac{\partial u_k}{\partial x_k} \\ \frac{\partial u_k}{\partial \varphi_k} \end{bmatrix} \tag{4.18}$$

$$\frac{\partial u_k}{\partial x_k} = \frac{\partial}{\partial x_k}\left[\frac{\sum_{j=1}^{49} w_j z_j}{\sum_{j=1}^{49} w_j}\right] = \sum_{j=1}^{49} w_j \frac{\partial z_j^N}{\partial x_k} \tag{4.19}$$

$$\frac{\partial u_k}{\partial \varphi_k} = \frac{\partial}{\partial \varphi_k}\left[\frac{\sum_{j=1}^{49} w_j z_j}{\sum_{j=1}^{49} w_j}\right] = \sum_{j=1}^{49} w_j \frac{\partial z_j^N}{\partial \varphi_k} \tag{4.20}$$

To compute the derivative of the normalized premises, the product rule has to be used.

$$\frac{\partial z_j^N}{\partial x_k} = \frac{\frac{\partial z_j}{\partial x_k} z_{sum} - z_j \frac{\partial z_{sum}}{\partial x_k}}{z_{sum}^2} \tag{4.21}$$

$$\frac{\partial z_j^N}{\partial \varphi_k} = \frac{\frac{\partial z_j}{\partial \varphi_k} z_{sum} - z_j \frac{\partial z_{sum}}{\partial \varphi_k}}{z_{sum}^2} \tag{4.22}$$

The derivatives of the unnormalized premises and their sum are

$$\begin{aligned} \frac{\partial z_j}{\partial x_k} = {} & \frac{\partial \mu_{x,1}}{\partial x_k}(\mu_{\varphi,1} w_1 + \mu_{\varphi,2} w_2 + \cdots + \mu_{\varphi,7} w_7) + \\ & \frac{\partial \mu_{x,2}}{\partial x_k}(\mu_{\varphi,1} w_8 + \mu_{\varphi,2} w_9 + \cdots + \mu_{\varphi,7} w_{14}) + \cdots \\ & \frac{\partial \mu_{x,7}}{\partial x_k}(\mu_{\varphi,1} w_{43} + \mu_{\varphi,2} w_{44} + \cdots + \mu_{\varphi,3} w_{49}) \end{aligned} \tag{4.23}$$

$$\begin{aligned} \frac{\partial z_j}{\partial \varphi_k} = {} & \frac{\partial \mu_{\varphi,1}}{\partial \varphi_k}(\mu_{x,1\,1} w + \mu_{x,2\,2} w + \cdots + \mu_{x,7\,7} w) + \\ & \frac{\partial \mu_{\varphi,2}}{\partial \varphi_k}(\mu_{x,1\,8} w + \mu_{x,2\,9} w + \cdots + m_{7\,1} w 4) + \cdots \\ & \frac{\partial \mu_{\varphi,7}}{\partial \varphi_k}(\mu_{x,1\,43} w + \mu_{x,2\,44} w + \cdots + \mu_{x,7\,49} w) \end{aligned} \tag{4.24}$$

$$\frac{\partial z_{sum}}{\partial x_k} = \sum_{j=1}^{4} 9 \frac{\partial z_j}{\partial x_k} \tag{4.25}$$

$$\frac{\partial z_{sum}}{\partial \varphi_k} = \sum_{j=1}^{4} 9 \frac{\partial z_j}{\partial \varphi_k} \tag{4.26}$$

The derivatives of the membership function to the input depend on the type of function used. For the sigmoid functions it is

$$\frac{\partial \mu_{x,i}}{\partial x_k} = -\frac{1}{a_i} \mu_{x,i}(x_k)(1 - \mu_{x,i}(x_k)) \tag{4.27}$$

and for the gaussian funtions it results in

$$\frac{\partial \mu_{x,i}}{\partial x_k} = \frac{2}{a_i^2}(x_k - c_i)\mu_{x,i}(x_k) \tag{4.28}$$

With all derivatives from the update equation broken down in terms which can be taken directly from the model or the controller, the change of consequence parameters can be calculated in every time step.

The derivatives to update the premise parameters are calculated analog except for 4.27 and 4.28, which are

$$\frac{\partial \mu_{x,i}}{\partial c_i} = -\frac{1}{a_i} \mu_{x,i}(x_k)(1 - \mu_{x,i}(x_k)) \tag{4.29}$$

$$\frac{\partial \mu_{x,i}}{\partial a_i} = -\frac{1}{a_i^2}(x_k - c_i)\mu_{x,i}(x_k)(1 - \mu_{x,i}(x_k)) \tag{4.30}$$

for sigmoid functions and

$$\frac{\partial \mu_{x,i}}{\partial c_i} = \frac{2}{a_i^2} \mu_{x,i}(x_k)(x_k - c_i) \tag{4.31}$$

$$\frac{\partial \mu_{x,i}}{\partial a_i} = \frac{1}{a_i^2} \mu_{x,i}(x_k)(x_k - c_i) \tag{4.32}$$

for gaussian functions.

In the navigation controller the premise parameters, as well as the consequence parameters can be chosen symmetrically, as the strategy to reach the desired position is mirror-inverted to the middle lane. Therefore in updating the parameters only one half is used and projected on the other half of the parameters. As the starting states are chosen to symmetrical, no information is lost.

The parameters of the slopes and centers had to be restricted. Especially the slopes would rise and lead to an overlapping of all membership function. That is there are maximum and minimum limts for components of the *a*- and *c*-vector.

The learning rates were desingned to be slightly adaptable. A simple system was chosen in which every parameter has its own learning rate. If the parameter is to be adapted in the current episode in the same direction (positive ornegative) as in the lastepisode, the

learning rate increasing with the factor 1.1 If the directions are different, the learning rate will be decreased by 0.5. For every parameter vector a maximal rate is defined to avoid that rates swing up fast and make the learning process instable. In general the learning rates had to be chosen very small.

Forthe training with dynamic a technique called "incremental learning" was employed. This means at the beginning of training starting positions close the the goal position (in x-position and angle) were used. When parameters for these were learned with success the starting positions were chosen to become more and more difficult. Accordingly, the algorithm 4.1 is to run numerous times with changed starting parameters for approximitaly 1000 episodes. This way it can be made sure the algorithm learns the correct steering technique, with the possibility to check after every learning stage.

1: Initialize premise parameter vectors a and c and vector with rule consequences w
2: Set symmetric starting points for training $s_{0,1}$ . . . $s_{0,M}$
3: for each episode i do
4: for each starting position j do
5: for each time step k do
6: Compute control signal $u_k = f_c(x_k, \varphi_k)$
7: Apply u_k to model and compute state in next time step $s_{k+1} = f_m(s_k, u_k)$
8: Caluclate error to desired position $e_k = s_k - s^*$
9: Compute total derivatives with equations 4.14 - 4.32
10: Calculate weight changes and them up

$$\Delta_{sum} w = \Delta_{sum} w + e_k \frac{\partial s_{k+1}}{\partial w}$$

$$\Delta_{sum} c = \Delta_{sum} c + e_k \frac{\partial s_{k+1}}{\partial c}$$

$$\Delta_{sum} a = \Delta_{sum} w + e_k \frac{\partial s_{k+1}}{\partial a}$$

11: end for
12: end for
13: Calculate changes of parameters

$$\Delta w = \frac{\Delta_{sum} w}{k_{total}}$$

$$\Delta c = \frac{\Delta_{sum} c}{k_{total}}$$

$$\Delta a = \frac{\Delta_{sum} a}{k_{total}}$$

14: Update step sizes
15: Update weights and premise parameters

$$w = w - \eta_w \Delta w$$

$$c = c - \eta_c \Delta c$$

$$a = a - \eta_a \Delta a$$

16: Restrict premise parameters and mirror all parameters
17: end for

Algorithm 4.1 – Training the neuro fuzzy navigation controller with dynamic backpropagation

4.3 Obstacle Avoidance

4.3.1 Structure of the Neuro Fuzzy Controller for Obstacle Avoidance

The idea was to design an additional neural fuzzy controller for obstacle avoidance. It generates a variable δ_{plus} which is added to the steering angle δ of the navigation controller. With this manipulation the steering angle is adpated if an obstacle is detected. A diagram of the control cycle with two controllers can be seen in figure 4.5. The

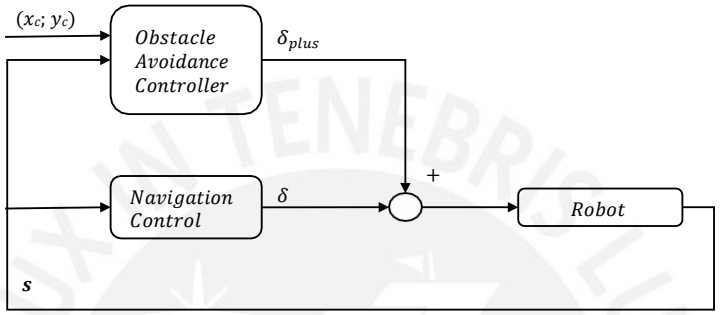


Figure 4.5 – Control cycle with Navigation and Obstacle Avoidance Controller

structure of the obstacle avoidance controller is again in 5 layers as described in 3.9 and can be seen in figure 4.6

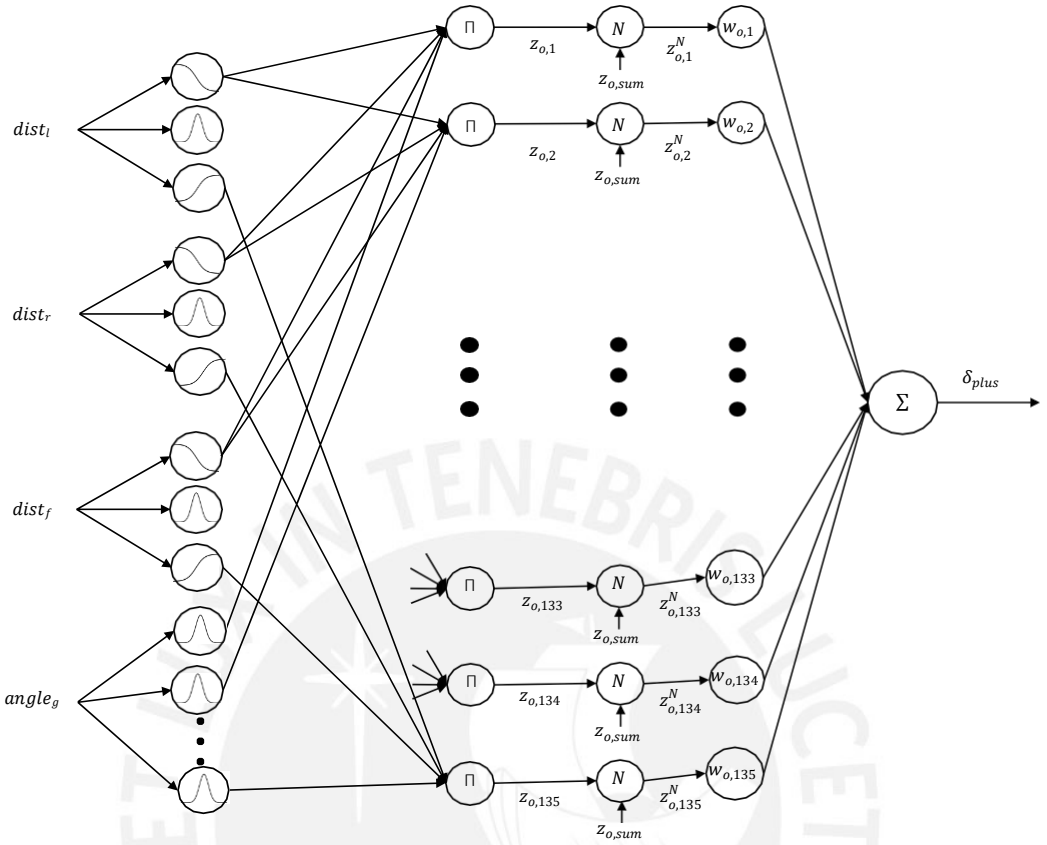


Figure 4.6 – Structure of the controller for obstacle avoidance

The input variables for the controller are the detected distance to an object on the left, front and right of the robot and the current orientation angle of the robot. The membership functions for the input can be seen in figure 4.7, the chosen parameter values and their liguistic interpretation in table 4.2. The output is the manipulation variable δ_{plus}, which corrects the steering angle from the navigation controller to avoid obstacles.

	c_l	a_l	linguistic interpretation	c_g	a_g	linguistic interpretation
1	1	1	TOO NEAR	$-\frac{3\pi}{4}$	0.3491	DOWN RIGHT
2	4	3	NEAR	$-\frac{\pi}{4}$	0.3491	UP RIGHT
3	7	0.5	FAR	0	0.3491	UP
4				$\frac{\pi}{4}$	0.34914	UP LEFT
5				$\frac{3\pi}{4}$	0.34914	DOWN LEFT

Table 4.2 – Initial center and slope parameters as well as linguistic interpretation of the membership functions for the obstacle avoidance control system. The parameters are the same for the three distance inputs.

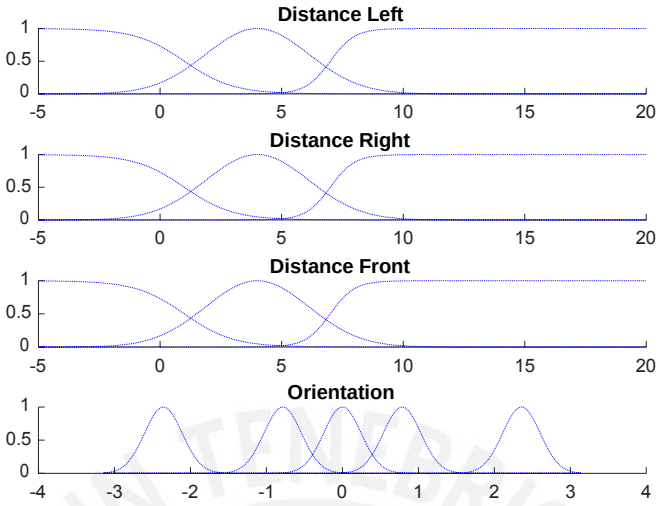


Figure 4.7 – Membership-Functions of the Obstacle Avoidance Algorithm

4.3.2 Simulation of Obstacle Recognition

The next step was to design a simulation for the recognition of obstacles, which would provide the distance between robot and object as if it was measured. Circular objects were chosen for this simulation because in that case the distance between robot and object is easily computable with the following equation

$$dist_c = \sqrt{(x_c - x)^2 + (y_c - y)^2} - R \tag{4.33}$$

where $(x_c;\ y_c)$ are the center coordinates of the cicular object and R is its radius. Additionally to the distance it needs to be decided if the obstacle is on the left, right or front of the robot. The angle to the center of the obstacle is determined with the equation

$$\beta_c = arctan\left(\frac{(y_c - y)}{x_c - x}\right) - \frac{\pi}{2} \tag{4.34}$$

The standard value for the three distance inputs is 100, symbolizing that the path is clear. This value is overwritten by the distance to the object if the angle to the object falls into the following ranges:

$-\frac{\pi}{2} < \beta_c < -\frac{\pi}{8}$ → Object left, $dist_l = dist_c$

$-\frac{\pi}{4} < \beta_c < \frac{\pi}{4}$ → Object front, $dist_f = dist_c$

$$\frac{\pi}{8} < \beta_c < \frac{\pi}{2} \quad \rightarrow \text{Object right, } dist_r = dist_c$$

The ranges are overlapping to recognize if an object is, for example, completely on the left or in front and slightly to the left (see figure 4.8).

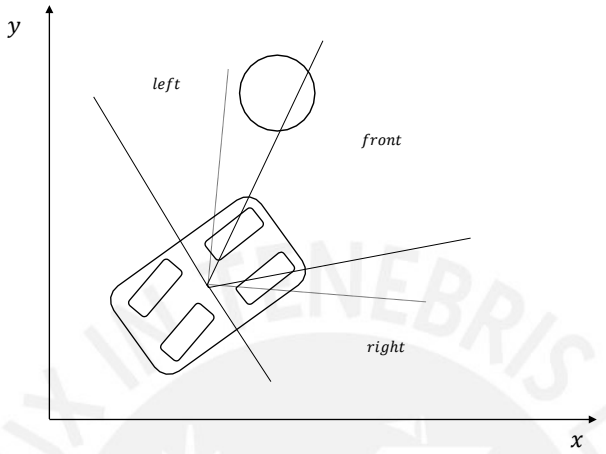


Figure 4.8 – Recognition of obstacles in tree sectors. The continious lines describe the left and right section, the dashed line the front section.

4.3.3 Training with Dynamic Backpropagation

Dynamic Backpropagation is once again employed to train the parameters of the obstacle avoidance controller. In the cost function it was important to combine a punshiment for coming too near to an obstacle and a reward for staying close to the direct path to the goal position.

$$J_o = \frac{1}{2}(x_k - x*)^2 + \frac{1}{2}(\varphi_x - \varphi*) - \frac{1}{2}(x_k - x_c)^2 - \frac{1}{2}(y_k - y_c)^2$$

With the distance to the obstacle being evaluated as negative, a greater distance to the obstacle comes with a minimalization, which is the desired outcome. For the obstacle avoidance the distance in y-direction is also considered, so it is added to the state vector

$$s_k = \begin{bmatrix} x_k \\ \varphi_k \\ y_k \end{bmatrix} \tag{4.35}$$

The parameter updates are computed with

$$\Delta w_{o,i} = \frac{\partial \overline{J_o}}{\partial w_{o,i}} \tag{4.36}$$

$$\Delta c_{o,i} = \frac{\partial \overline{J_o}}{\partial c_{o,i}} \tag{4.37}$$

$$\Delta a_{o,i} = \frac{\partial \overline{J_o}}{\partial a_{o,i}} \tag{4.38}$$

The detailed derivation of the update equations works similar to the derivation in chapter 4.2.2 and can be found in the appendix. The employed algorithm has the same structure as algorithm 4.1.

4.3.4 Training with Actor-Critic-Algorithm

Another training concept is applied the obstacle avoidance controller to compare the results afterwards. An actor-critic training algortihm based on the explanation in 3.2.4 was implemented to adapt the parameters. The final algorithm can be seen in algorithm 4.2 and is further explained below. To present the algorithm clearer premise and consequence parameters are concluded in the variable θ_o. Note, that in the critic networkonly the consequence parametersare updated. The premise parameters should be the same as in the control network so every action can be critisized in relation to the state premises that led to it. In the beginning the all parameters are initialized. The premise parameters of the critic, target and critic target network were chosen to be the same as the ones from the controller network. The critic and the critic target network have the taken action as an additional input. The membership function chosen for this input is shown in figure 4.9. The concequence parameters were chosen to be 0 inall networks. One episode in this training consists of only one starting position and the resulting path in 1350 time steps. The starting state for every episode is determined randomly. At the beginning of each episode, the Ornstein-Uhlenbeck process layered upon the controller output to realize explorational behaviour is initilized. In every timestep k the current state is given to the controller and the correction variable δ_{plus} is calculated. The noice of the Ornstein-Uhlenbeck process is added and the result manipulates the steering angle from the navigation controller. The new steering angle is the control input for the kinematic model and the following state s_{k+1} is computed. Depending on that state the system is getting a reward. The reward r was chosen as follows: If the difference between the desired position x^* and the position at the timestep $k+1$ issmaller than 0.1 the reward is a positive 1. Everywhere else the reward

Initialize parameters $\boldsymbol{\theta}_o$, $\boldsymbol{\theta}_{o,c}$,$\boldsymbol{\theta}_o{}^t$, $\boldsymbol{\theta}_{o,c}{}^t$
for every episode j do
  Initilize random exploration noise process $\mathcal{N}$
  Randomly select a starting position s_0
  for every time step k do
    Determine the taken action $a_k = \delta_{plus}(s_k) + \mathcal{N}$
    Apply action to model and compute next time step $s_{k+1} = f_m(s_k, a_k)$
    Get reward $r = r(s_{k+1})$
    Store transition (s_k, s_{k+1}, a_k, r) in replay memory
    if size of replay memory > n then
      Randomly sample a minibatch of n transistions from replay memory
      for every samply in minibatch i do
        Compute output of critic network Q
        Compute output of target network δ^t_{plus}
        Compute output of critic target network Q^t
      end for
      Update critic parameters

$$w_{o,c} = w_{o,c} + \eta_{o,c} \frac{\partial L(w_{o,c})}{\partial w_{o,c}}$$
$$= w_{o,c} + \eta_{o,c} (r + \gamma Q^t(s_{k+1}, a_{k+1})) \frac{\partial Q}{\partial w_{o,c}}$$

      Update actor parameters: $\boldsymbol{\theta}_o = \boldsymbol{\theta}_o + \eta_o \frac{\partial Q(s_k, a_k, \boldsymbol{\theta}_o)}{\partial \boldsymbol{\theta}_o}$
      Update target critic parameters: $\boldsymbol{\theta}_{o,c}{}^t = \tau \boldsymbol{\theta}_{o,c} + (1 - \tau)\boldsymbol{\theta}_{o,c}{}^t$
      Update target parameters: $\boldsymbol{\theta}_o{}^t = \tau \boldsymbol{\theta}_o + (1 - \tau)\boldsymbol{\theta}_o{}^t$
    end if
  end for
end for

Algorithm 4.2 – Actor-Critic-Learning for the Obstacle-Avoidance Controller

is bigger the closer the robot is to the ideal position and the reward is calculated with

$$r = \frac{0.1}{\sqrt{(x_{k+1} - x^*)^2}} \tag{4.39}$$

This helps the result to stay close to the desired position. The control system is punished if the robot is driving in the wrong direction (in negativ y direction) or it is getting too close to the obstacle

$$dist_{TOOCLOSE} = -\sqrt{(x_{k+1} - x_c)^2 + (y_{k+1} - y_c)^2} - R - 0.5 \tag{4.40}$$

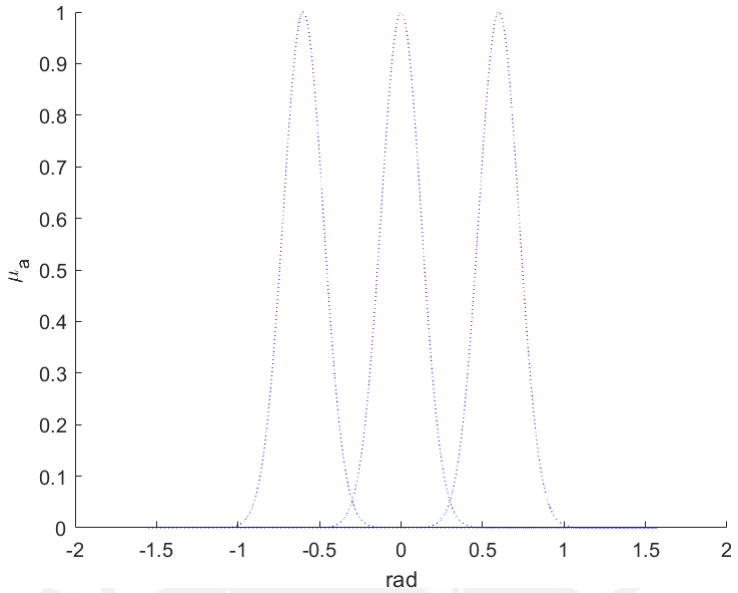


Figure 4.9 – Membership function of the additional input of the critic networks

In both cases the reward is reduced to -1. To be used in training the state s_k, the taken action δ_{plus}, the gained reward r and the newly reached state s_{k+1} are saved in a replay memory. This algorithm learns with the transitions in the replay memory to improve the quality of learning. The minimum size of the replay memory to start learning was chosen to be 300. This means in the first 300 time steps of an episode the network is not trained at all but only collects transition samples. As soon as the replay memory has enough entries a minibatch of $n = 100$ transitions is sampled from it and used for training. At first the critic network determines the critic of the action taken at time step k. Then the target network determines the action the network would take in the time step $k+1$ with the current parameters. The critic target network critizises this action. With the outputs of these networks the parameter updates can be calculated. The critics network parameters are adapted with

$$w_{o,c} = w_{o,c} + \eta_{o,c} \frac{\partial L(w_{o,c})}{\partial w_{o,c}} \tag{4.41}$$

$$= (r_k + \gamma Q^t(s_{k+1}, a_{k+1})) \cdot \frac{\partial Q}{\partial w_c} \tag{4.42}$$

where u_c is the output of the critic network and $\frac{\partial Q}{\partial u_c}$ is the vector of the critics normalized premise parameter z_c^N.

The controllers parameters are updated with

$$\boldsymbol{\theta}_o = \boldsymbol{\theta}_o + \eta_o \frac{1}{n} \sum_{i=1}^{n} \frac{\partial Q(s_k, u_k, \boldsymbol{\theta}_o)}{\partial \boldsymbol{\theta}_o} \tag{4.43}$$

$$= \boldsymbol{\theta}_o + \eta_o \frac{1}{n} \sum_{i=1}^{n} \frac{\partial Q(s_k, u_k, \boldsymbol{\theta}_o)}{\partial u_k} \frac{\partial u_k}{\partial \boldsymbol{\theta}_o} \tag{4.44}$$

$$\tag{4.45}$$

The last two derivatives are computed this way:

$$\frac{\partial Q(s_k, u_k, \boldsymbol{w}_o)}{\partial u_k} = \boldsymbol{\theta}_o \frac{\partial z_{o,c}^N}{\partial u_k} \tag{4.46}$$

$$= \boldsymbol{\theta}_o \frac{\frac{\partial z_{o,c}}{\partial u_k} z_{o,sum} - z_{o,c} \frac{\partial z_{o,sum}}{\partial u_k}}{z_{o,sum}^2} \tag{4.47}$$

$$\frac{\partial u_k}{\partial \boldsymbol{\theta}_o} = z^N \tag{4.48}$$

Chapter 5

Results

5.1 Results of the Navigation Controller

The newly added training of premise parameters adapted the inital membership functions (figure 4.4) to the ones shown in figure 5.1. The membership functions of the position in

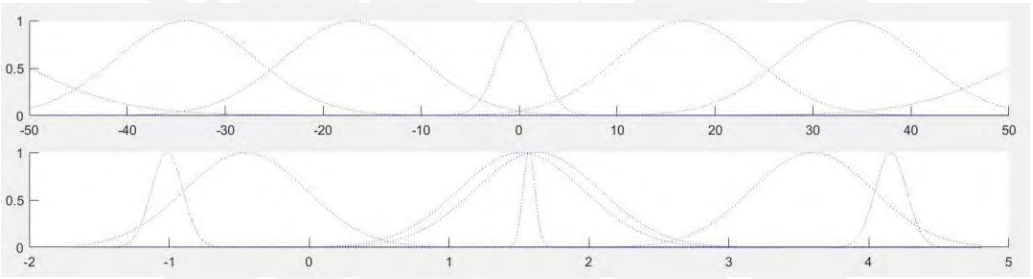
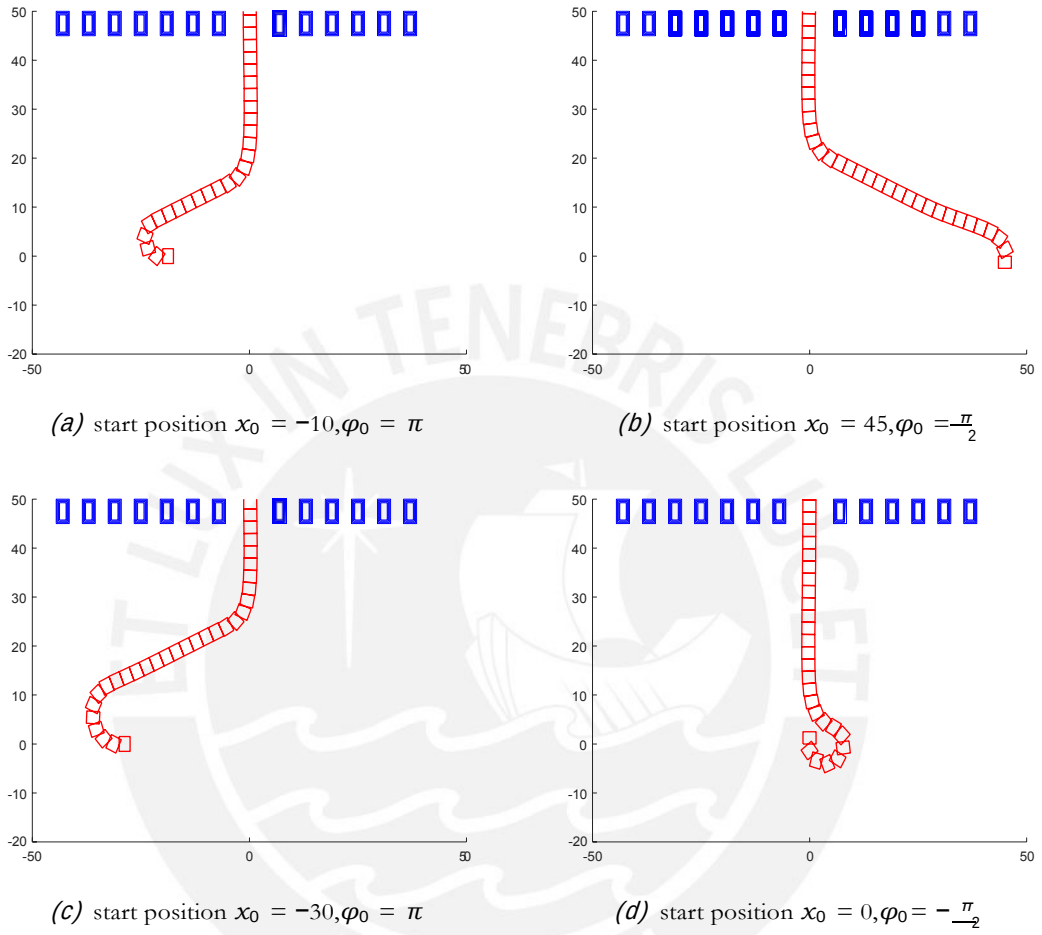


Figure 5.1 – Membership functions of the x-position (above) and the orientation (below) after training

x- direction were barely changed. Only the one representing ZERO is now less wide as before. This may increases the accuracy at the goal position, because the range in which the robot is considered to be at the desired position is now smaller. On the contrary to the position, the membership functions of the orientation changed a lot and lost a bit of their linguistic interpretability. However, it is supposed that by moving the membership functions for UP LEFT and UP RIGHT closer together in the center the accuracy was once again increased, because now even small variations from the ideal $\varphi^* = \frac{\pi}{2}$ can be detected and corrected.

The neuro fuzzy navigation controller obtained by training premise and consequence parameters with dynamic backpropagation was succeccfully tested in a simulation for different starting states. No matter which starting position and orientation was chosen,

the robot would always find the desired position in a fast way. A few examples of taken paths for different starting states are shown in the following figures 5.2. Even in figure



(a) start position $x_0 = -10, \varphi_0 = \pi$ *(b)* start position $x_0 = 45, \varphi_0 = -\frac{\pi}{2}$

(c) start position $x_0 = -30, \varphi_0 = \pi$ *(d)* start position $x_0 = 0, \varphi_0 = -\frac{\pi}{2}$

Figure 5.2 – Paths taken by the robot to parking position for different start positions

5.2 b, where the starting x-value was chosen very close to the limits of the operating range, the robot could reach the goal position in under 50 units in the y-direction. It can also be observed, that the robot uses the strategy mentioned in chapter 4.2.1 and drives to the center, while correcting its orientation, before driving in a straight line to the parking spot.

By changing the desired state the robot was also able to approach different parking spots. This can be seen in the figures below.
Another test scenario was to let the robot navigate the way to a circular path and follow

it. To realize this the desired position values were chosen to be:

$$x^* = x_c - (x - x_c)\frac{R}{\sqrt{((x_c - x_k)^2 + y_k^2)}} \tag{5.1}$$

$$\varphi^* = -\arctan\left(\frac{y_k}{x_c - x_k}\right) \tag{5.2}$$

where $(x_c;\ y_c)$ are the center coordinates of the circle, R the radius of the circle and L the length of the robot. Once the robot reaches the circle the steering angle only needs to stay constant at a certain degree to stay on the path. This angle can be determined and is added to the steering angle from the controller as a 'correction term'.

$$u_{k,corrected} = u_k + \frac{\pi}{2} - \arctan\left(\frac{R}{L}\right); \tag{5.3}$$

In figure 5.3 the path for reaching and following a circular path is shown.

The final test scenario was to let the robot follow a sinusodial path. The path is predefined with the parameters amplitude A, frequence f. The desired state for this case is

$$x^* = A \cdot cos(f \cdot y_k) \tag{5.4}$$

$$\varphi^* = arctan(A \cdot f \cdot sin(f \cdot y_k)) \tag{5.5}$$

Againa correction term is needed to smooth the robots motion while following the path.

$$u_{k,corrected} = u_k + (f^2 \cdot A \cdot sin(f \cdot x_k)) \cdot \frac{L}{r \cdot sec(f)^2}; \tag{5.6}$$

The robot was able to reach the sinusodial path and follow it, which can be seen in figure 5.4 below. All in all the navigation algorithm could could reach very good results in simulation. The robot was able to reach any desired parking position from an arbitraty start state and could follow circular and sinusodial shapes lines.

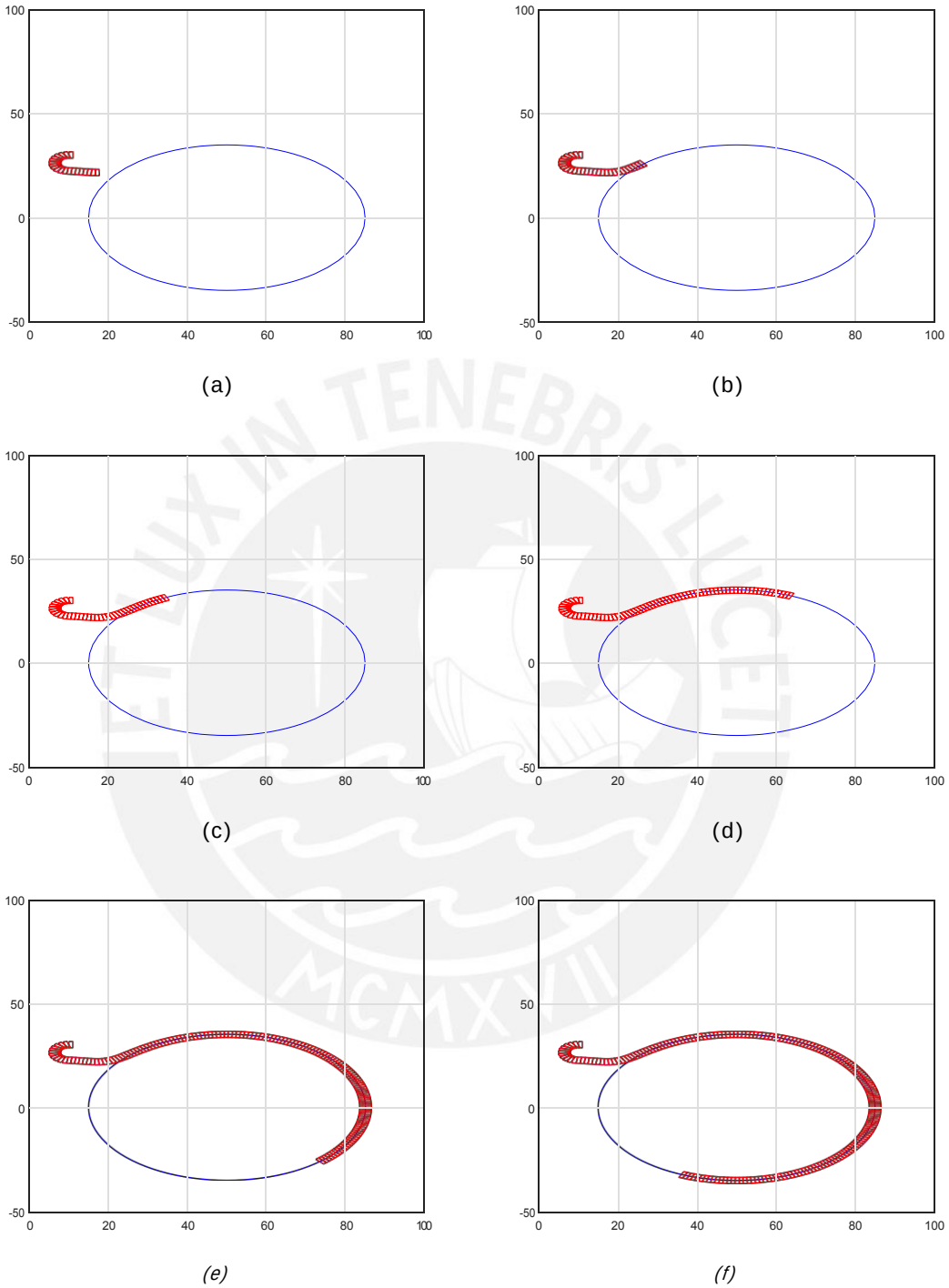


Figure 5.3 – Robot finding and following a circular path. The circle has the parameters $(x_c;\ y_c) = (50;\ 0)$, $R = 35$ and the starting state is $(x_0;\ y_0;\ \varphi_0) = (10;\ 30;\ \pi)$

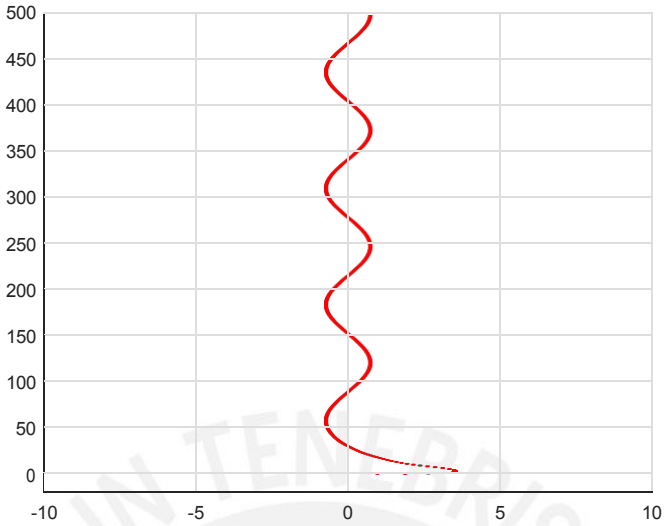


Figure 5.4 – Following a sinusodial path.

5.2 Results of the Obstacle Avoidance Controller

At first the results of the obstacle avoidance controller trained with dynamic backpropagation are presented. This method proved to be more difficult to apply to the obstacle avoidance problem than to the navigation problem before. The reason is that the definition of 'easy' cases to start with the incremental learning is not as clear as in the navigation case. Choosing starting points at the side of the obstacle in the beginning and moving to the center seemed to be a good possibility but did not always result in the desired paths. The controller ended up navigating the robot around the right side of the obstacle for most starting positions. The results can bee seen in the following figures. The robot can avoid the obstacle but often not on the ideal path. The controller trained with deterministic policy gradient could not reach these results. Althought seeming to get in the right direction an collision was not always avoided as can be seen in the following figures

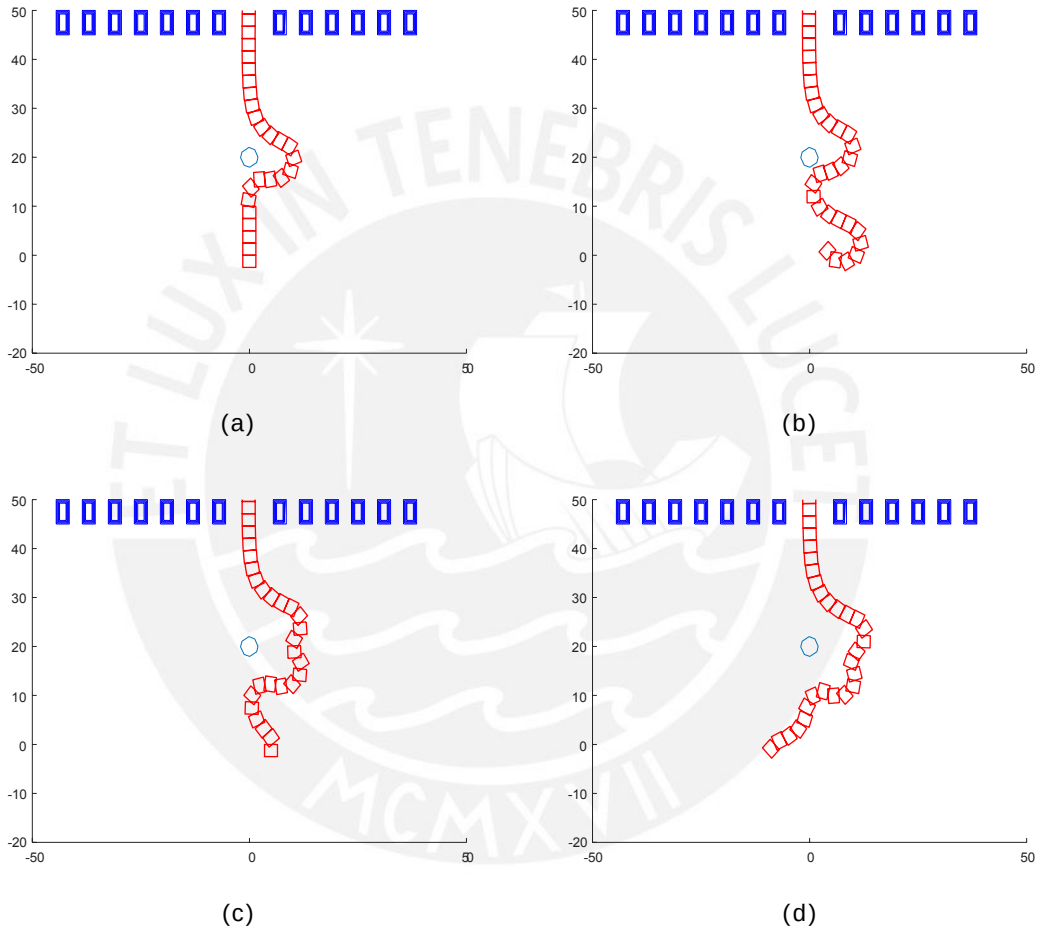


Figure 5.5 – Results of the obstacle avoidance controller trained with dynamic backpropagation

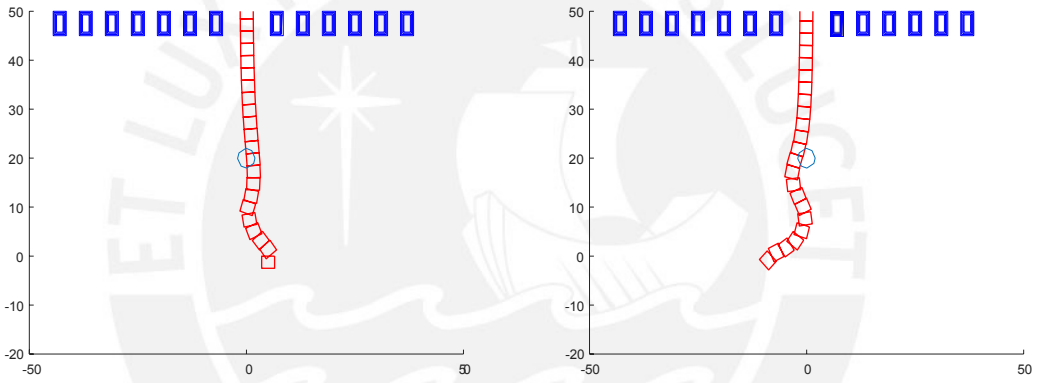


Figure 5.6 – Failed results of the obstacle avoidance controller trained with deterministic policy gradient

Chapter 6

Conclusion

The thesis focused on the development of a neuro-fuzzy controller for a 4-wheeled car-type robot to solve a parking problem while avoiding obstacles.
At first the navigation to a desired position on a clear path was considered. The neural fuzzy navigation controller for that problem had been set up in an earlier work and has a zero-order Takagi-Sugeno structure. Until now it had only been trained in its consequence parameters and was expanded in this thesis by the dynamic backpropagation training for the premise parameters.
In the second part of the work a controller for avoiding obstacles was developed. This controller returns a manipulation value, which is added to the output of the navigation controller should an object come near. It was trained with two different neural methods, which are called 'Dynamic Backpropagation' and 'Deterministic Policy Gradient learning'.
For the navigation task the controller trained with dynamic backpropagation led to very good results. The robot was able to reach the desired parking position without problems. With the algorithm it was also possible to let the robot drive in circular shapes and sinsusodial paths.
The obstacle avoidance problem is still in need of further developing. The controller trained with dynamic backpropagation was able to avoid obstacle but increased its path to much while doing this. The controller trained with deterministic policy gradient seemed to have the opposite problem - a collision-free navigation cannot be guaranteed for the whole range of x-values.
In future works the obstacle avoidance needs to be developed further. Maybe more membership function could help the case to get asmoother trajectory aroundtheobjects. The controller trained with deterministic policy gradient seems to need more training steps. Maybe a better, adaptable step size could solve this problem.

List of Figures

List of Tables

Bibliography

[1] Cardenas, Antonio M. ; Razuri, Javier G. ; Sundgren, David ; Rahmani, Rahim: Autonomous Motion of Mobile Robot Using Fuzzy-Neural Networks. In: Castro, Félix (Hrsg.): *12th Mexican International Conference on Artificial Intelligence (MICAI 2013)*. Piscataway, NJ : IEEE, 2013. – ISBN 978–1–4799–2605–3, S. 80–84

[2] Zadeh, L. A.: Fuzzy sets. In: *Information and Control* 8 (1965), Nr. 3, S. 338–353. http://dx.doi.org/10.1016/S0019-9958(65)90241-X. – DOI 10.1016/S0019–9958(65)90241–X. – ISSN 00199958

[3] Ross, Timothy J.: *Fuzzy Logic with Engineering Applications*. 4. Aufl. s.l. : Wiley, 2016 http://gbv.eblib.com/patron/FullRecord.aspx?p=4694618. – ISBN 1119235863

[4] Czogała, Ernest ; Łęski, Jacek: *Studies in Fuzziness and Soft Computing*. Bd. 47: *Fuzzy and Neuro-Fuzzy Intelligent Systems*. Heidelberg : Physica-Verlag HD, 2000. http://dx.doi.org/10.1007/978-3-7908-1853-6. http://dx.doi.org/10.1007/978-3-7908-1853-6. – ISBN 9783662003893

[5] Narendra, Kumpati S. ; Parthasarathy, Kannan: Neural networks and dynamical systems. In: *International Journal of Approximate Reasoning* 6 (1992), Nr. 2, S. 109–131. http://dx.doi.org/10.1016/0888-613X(92)90014-Q. – DOI 10.1016/0888–613X(92)90014–Q

[6] Sutton, Richard S. ; Barto, Andrew G.: *Reinforcement learning: An introduction*. 1998 (Adaptive computation and machine learning). – ISBN 978–0–262–19398–6

[7] Lillicrap, Timothy P. ; Hunt, Jonathan J. ; Pritzel, Alexander ; Heess, Nicolas ; Erez, Tom ; Tassa, Yuval ; Silver, David ; Wierstra, Daan: Continuous control with deep reinforcement learning. In: *CoRR* abs/1509.02971 (2015)

[8] Silver, David ; Lever, Guy ; Heess, Nicolas ; Degris, Thomas ; Wierstra, Daan ; Riedmiller, Martin: Deterministic Policy Gradient Algorithms. In: *ICML*. Beijing, China, 2014

[9] Godjevac, J. ; Steele, N.: Adaptive neuro-fuzzy controller for navigation of mobile robot. In: *International Symposium on Neuro-Fuzzy Systems*. Lausanne, Switzerland : Repro EPFL, 1997. – ISBN 0–7803–3367–5, S. 111–119

[10] Gui-juan, Wang ; Zuo-xun, Wang ; Yan-rong, Wu ; Hong-dong, Xu: The Application of Neuro-Fuzzy Controller in the Washing Machine Control System. In: *Second International Conference on Intelligent Computation Technology and Automation, 2009*. Piscataway, NJ : IEEE, 2009. – ISBN 978–0–7695–3804–4, S. 818–821

[11] Jurado, Francisco ; Ortega, Manuel ; Cano, Antonio ; Carpio, José: Neuro-fuzzy controller for gas turbine in biomass-based electric power plant. In: *Electric Power Systems Research* 60 (2002), Nr. 3, S. 123–135. http://dx.doi.org/10.1016/S0378-7796(01)00187-0. – DOI 10.1016/S0378–7796(01)00187–0. – ISSN 03787796

[12] Budiharto, Widodo ; Jazidie, Achmad ; Purwanto, Djoko: Indoor Navigation Using Adaptive Neuro Fuzzy Controller for Servant Robot. In: *Second International Conference on Computer Engineering and Applications (ICCEA), 2010*. Piscataway, NJ : IEEE, 2010. – ISBN 978–1–4244–6079–3, S. 582–586

[13] Zhu, Anmin ; Yang, Simon X.: Neurofuzzy-Based Approach to Mobile Robot Navigation in Unknown Environments. In: *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)* 37 (2007), Nr. 4, S. 610–621. http://dx.doi.org/10.1109/TSMCC.2007.897499. – DOI 10.1109/TSMCC.2007.897499. – ISSN 10946977

[14] Rusu, P. ; Petriu, E. M. ; Whalen, T. E. ; Cornell, A. ; Spoelder, H.J.W.: Behavior-based neuro-fuzzy controller for mobile robot navigation. In: *IEEE Transactions on Instrumentation and Measurement* 52 (2003), Nr. 4, S. 1335–1340. http://dx.doi.org/10.1109/TIM.2003.816846. – DOI 10.1109/TIM.2003.816846. – ISSN 0018–9456

[15] Kuremoto, Takashi ; Tsubaki, Kazuhiro ; Obayashi, Masanao ; Mabu, Shingo ; Kobayashi, Kunikazu: A neuro-fuzzy reinforcement learning system for autonomous robot dealing with continuous space. In: *NCSP 2016 : 2016 RISP*

International Workshop on Nonlinear Circuits, Communications and Signal Processing, Hawaii, United States of America, Mar 06-Mar 09,2016

[16] Al-Dabooni, Seaar ; Wunsch, Donald: Mobile robot control based on hybrid neuro-fuzzy value gradient reinforcement learning. In: *IJCNN 2017*. Piscataway, NJ : IEEE, 2017. – ISBN 978–1–5090–6182–2, S. 2820–2827

[17] Brahimi, Somia ; Azouaoui, Ouahiba ; Loudini, Malik: Neuro-Fuzzy Navigation for a Car-Like Robot in Unknown Environments, 2015

Appendices

Appendix A

The derivation of update equations for the parameters of the obstacle avoidance controller is shown exemplary for a concequence parameter w_i, but is, except for the derivatives of the membership functions analog for the premise parameters.

$$w_{o,i} = w_{o,i} - \eta_{o,i} \frac{\sum_{k=1}^{k_{total}} \Delta w_{o,i,k}}{k_{total}} \tag{.1}$$

$$\Delta w_{o,i} = \frac{\partial J_o}{\partial w_{o,i}} \tag{.2}$$

With the cost function being

$$J_o = \frac{1}{2}(x_k - x*)^2 + \frac{1}{2}(\varphi_x - \varphi*)^2 + \frac{1}{2}(x_k - x_c)^2 + \frac{1}{2}(y_k - y_c)^2 \tag{.3}$$

the total derivative can be split up to

$$\frac{\partial J_o}{\partial w_{o,i}} = ((x_k - x*) + (x_c - x_k))\frac{\partial x_k}{\partial w_{o,i}} + (\varphi_k - \varphi*)\frac{\partial \varphi_k}{\partial w_{o,i}} + (y_c - y_k)\frac{\partial y_k}{\partial w_{o,i}} \tag{.4}$$

The total derivatives of the state variables can be calculated with the recursive formular described in chapter 3.2.3.

$$\frac{\partial s_{k+1}}{\partial w_{o,i}} = \frac{\partial s_{k+1}}{\partial w_{o,i}} + \frac{\partial s_{k+1}}{\partial s_k} + \frac{\partial s_{k+1}}{\partial u_k}\frac{\partial u_k}{\partial s_k}\frac{\partial s_k}{\partial w_{o,i}} \tag{.5}$$

(.6)

By applying the chain rule, the first derivative becomes

$$\frac{\partial s_{k+1}}{\partial w_{o,i}} = \frac{\partial s_{k+1}}{\partial u_k} \frac{\partial u_k}{\partial w_{o,i}} \tag{.7}$$

$$= \begin{bmatrix} 0 \\ -\frac{r}{L} \\ 0 \end{bmatrix} z_o^{N_i} \tag{.8}$$

The jacobian matrix of the system is derived from the model equations, listed in chapter 4.1

$$\frac{\partial s_{k+1}}{\partial s_k} = \begin{bmatrix} 1 & -r \cdot sin(\varphi_k) & 0 \\ 0 & 1 & 0 \\ 0 & r \cdot cos(\varphi_k) & 1 \end{bmatrix} \tag{.9}$$

The next two derivatives can be written as

$$\frac{\partial s_{k+1}}{\partial u_k} \frac{\partial u_k}{\partial s_k} = \begin{bmatrix} 0 \\ -\frac{r}{L} \\ 0 \end{bmatrix} \begin{bmatrix} \frac{\partial u_k}{\partial x_k} & \frac{\partial u_k}{\partial \varphi_k} & \frac{\partial u_k}{\partial y_k} \end{bmatrix} \tag{.10}$$

(.11)

$$\frac{\partial u_k}{\partial x_k} = \frac{\partial}{\partial x_k} \left[\sum_{j=1}^{N} w_{o,j} z_{o,j} \right] = \sum_{j=1}^{N} w_{o,j} \frac{\partial z_{o,j}^{N_j}}{\partial x_k} \tag{.12}$$

$$\frac{\partial u_k}{\partial y_k} = \frac{\partial}{\partial y_k} \left[\sum_{j=1}^{N} w_{o,j} z_{o,j} \right] = \sum_{j=1}^{N} w_{o,j} \frac{\partial z_{o,j}^{N_j}}{\partial y_k} \tag{.13}$$

$$\frac{\partial u_k}{\partial \varphi_k} = \frac{\partial}{\partial \varphi_k} \left[\sum_{j=1}^{N} w_{o,j} z_{o,j} \right] = \sum_{j=1}^{N} w_{o,j} \frac{\partial z_{o,j}^{N_j}}{\partial \varphi_k} \tag{.14}$$

The derivative of a normalized premise parameter z^N to $z_{o,i}$ the inputs will be done exemplary to the input x_k

$$\frac{\partial z_{o,i}}{\partial x_k} = \frac{\frac{\partial z_{o,i}}{\partial x_k} z_{o,sum} - z_{o,i} \frac{\partial z_{o,sum}}{\partial x_k}}{z_{o,sum}^2} \tag{.15}$$

$$\frac{\partial z_{o,i}}{\partial x_k} = \frac{\partial dist_l}{\partial x_k} \mu_r(dist_r) + \mu_l(dist_l) \frac{\partial dist_r}{\partial x_k} \mu_f(dist_f) \tag{.16}$$

$$+ \mu_l(dist_l) \mu_r(dist_r) \frac{\partial dist_f}{\partial x_k} \mu_g(goal) \tag{.17}$$

$$+ \mu_l(dist_l) \mu_r(dist_r) \mu_f(dist_f) \frac{\partial goal}{\partial x_k} \tag{.18}$$

$$\frac{\partial dist_l}{\partial x_k} = \frac{\partial dist_r}{\partial x_k} = \frac{\partial dist_f}{\partial x_k} = \frac{\partial dist_c}{\partial x_k} = \frac{(x_k - x_c)}{\sqrt{(x_k - x_c)^2 + (y_k - y_c)^2}} \tag{.19}$$

$$\frac{\partial goal}{\partial x_k} = \frac{2 y_k}{x_k^2 + y_k} \tag{.20}$$

$$\frac{\partial z_{o,sum}}{\partial x_k} = \sum_{i=1}^{81} \frac{\partial z_{o,i}}{\partial x_k} \tag{.21}$$