SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition and Intelligence

A Spiking Neural Network Trained with R-STDP for Person Following Using FMCW Data

Finn Capelle

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition and Intelligence

A Spiking Neural Network Trained with R-STDP for Person Following Using FMCW Data

Gepulste Neuronale Netze Trainiert mit R-STDP für Personen Verfolgung mit FMCW Radar Daten

Author:	Finn Capelle
Supervisor:	Prof. Dr.-Ing. Alois Knoll
Advisor:	M.Sc. Robin Dietrich
Submission Date:	15.11.2022

I confirm that this master's thesis in robotics, cognition and intelligence is my own work and I have documented all sources and material used.

Munich, 15.11.2022 Finn Capelle

Acknowledgments

Abstract

Robots are supporting us humans in many different areas where they need to be able to drive in unseen environments, for which they must follow lines or other objects to fulfil their functionality. For that they depend on robust and energy efficient driving algorithms to work as long as possible without needing to recharge. One specific area of application is person following, where they can support humans by carrying heavy objects for example. While learning based algorithms often show promising results, they mostly depend on ANNs, which have the disadvantage, that they are energy inefficient and computing intensive. In contrast, Spiking Neural Networks, which are more biological plausible, consume less energy as they work asynchronous. This makes them especially attractive in fields where systems depend on batteries.

This thesis introduces a SNN, that controls a robot for person following. The network is being trained with R-STDP, which is a biologically motivated way of training SNNs. In contrast, other SNNs build for following tasks are often tuned by hand. As input, the network receives a tracking signal from a CANN, which has been implemented in another thesis. In combination they present and end-to-end approach with SNNs for a person following robot. The network can react to different velocities of the person and follow arbitrary paths. Different test scenarios have been created and their results presented to show the capabilities of the proposed architecture.

Contents

1 Introduction

Robots are supporting us humans in many different areas. Often they carry out precise work, like producing computers or especially heavy work like assembling cars. These robots are mostly stationary and do not need the ability to move in unseen environments. But many other applications are dependent on exactly that ability. For that they need to be able to perceive their environment, to recognize obstacles or other objects which they can follow. Self-driving cars for example must recognize lines on the road, to stay within their lane and autonomous robots in warehouses need to follow lines or persons to assist in transporting goods. To follow these objects driving algorithms need to be robust and energy efficient, as these systems often rely on batteries.

To construct these driving algorithms learning based approaches often show very good results. In general, they are able to adapt to multiple problems and ANNs are used for numerous different tasks like object recognition and text recognition and translation. Depending on the use case different kinds of ANNs exist, which have been optimized for the particular problem. But most of them are still far from the functionality of biological neural networks and they are energy expensive, which makes them not ideal for use in person following, especially because these driving algorithms need to be constantly active. A computer performing image recognition among 1000 classes for example consumes about 250W. In contrast the human brain consumes only about 20W performing numerous actions like simultaneous recognition, reasoning, control and movement [RJP19]. Spiking Neural Networks try to overcome these problems, by being biologically more plausible, which makes them more energy efficient. For that reason they are used in this thesis to construct the driving algorithm for a person following robot.

1.1 Motivation

In recent years Spiking Neural Networks (SNNs), which are considered to be the 3. generation of ANNs, have become more popular in research. They mimic the processes of biological neurons and synapses more closely than 2. generation ANNs. Like

biological Neural Networks, they work asynchronous and propagate spikes, which makes them more energy efficient and possibly more powerful.

These advantages make it interesting to test how SNNs perform on mobile vehicles, with the example of person following. Even though the case is quite specific, it should be transferable to other following scenarios.

1.2 Problem Definition

To construct a person following robot, a spiking neural network will be created in this thesis to control the robot. It will be combined with a Spiking Continuous Attractor Network (CANN), that will provide tracking information. This will result in an end-to-end system for a person following robot, that solely relies on SNNs.

The robot must be able to follow the person with a constant distance. For that it must be able to adapt its velocity according to the tracked person and follow arbitrary paths, but it is assumed, that the robot does not need to navigate around obstacles.

1.3 State of the Art

In general, three different approaches exist today for person or object following robots. Most commonly systems provide the position of the object which is being tracked and from here on out an algorithm exists, that computes left and right wheel velocities or a steering angle and a velocity to follow that object. Very often the computations for that are done with a PID-Controller [SKC99; CST17; CC18], or they came up with their own set of functions [SM09; Miu+10; Jia+13].

Another option is to use an ANN, that takes images as input and predicts a predefined set of actions to follow an object or person in the input image [Pan+20; DM21].

SNNs often predict the velocity of the left and right wheel in form of spikes, which then need to be interpreted to send the actual velocity to the robot [Bin+18]. Weights for the SNNs are often set by hand [Wan+09; Kai+16].

1.4 Objective

The objective of this thesis is to construct and evaluate a person following robot, that solely works with SNNs and is already trained and constructed as such. In literature this is often not the case, frequently it can be found, that ANNs are converted to SNNs or that weights are set by hand.

The training in this thesis should be done with reinforcement learning, for which in SNNs typically R-STDP is used.

The robot will be evaluated in different following scenarios, like a random walk of the person or the person walking in small circles and tested for robustness against random noise spikes.

1.5 Approach

To follow a person a spiking neural network is constructed, that outputs commands for a robot to follow that person. The network will be trained using R-STDP, which is, to some extent, a biological plausible method [Izh07] of reinforcement learning. The robot will operate in a simulation, which makes it saver, cheaper and easier to implement the training process, as training data can easily be generated.

The network receives a tracking signal, from a Continuous Attractor Neural Network (CANN), which are a popular method of modeling continuous neural processes [Wu+16]. The CANN, which is used for this thesis, is a result from another master thesis [Aks22] and receives as input FMCW radar data.

1.6 Overview

The next chapter gives a brief overview of biological neurons and the history of neural networks. Also, the theory and fundamentals behind Spiking Neural Networks are explained. Chapter 3 describes other approaches for person following and highlights the most common ones. Even though no papers on person following with SNNs could be found, some papers are introduced, that follow lines, walls or try to reach a goal. Chapter 4 describes our solution to the problem of person following. First the input information the network receives is explained, to better understand the network architecture, which is introduced next. Afterwards the coding strategies for the network are explained and then the reward functions, which are needed for the section that goes into detail on how the network is trained. Chapter 5 evaluates the proposed solution with multiple scenarios and chapter 6 concludes the thesis, presents limitations and future work possibilities.

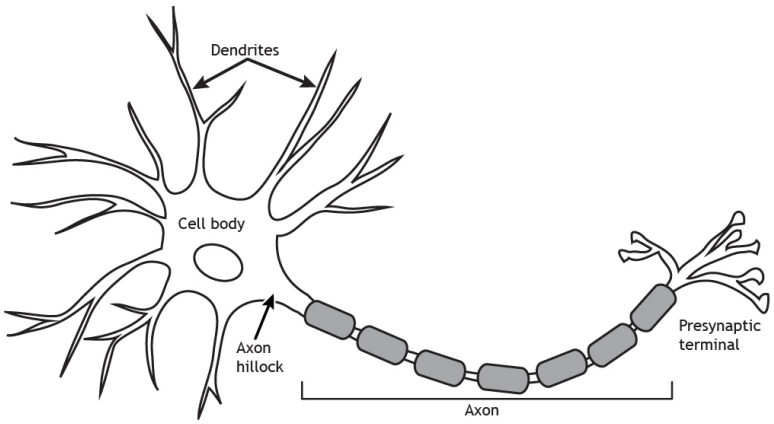


Figure 2.1: Overview of a Neuron. The Dendrites receive spikes from postsynaptic neurons, which increases the neurons potential. If the potential passes a certain threshold, a spike will be propagated along the Axon, that connects it to other neurons. Picture is taken from [20b].

2 Background

This chapter describes the relevant background information for this thesis. First a biological description of a neuron is given and afterwards a brief history of Neural Networks. The last part consists of the main concepts of Spiking Neural Networks which are needed to understand the following chapters.

2.1 Biological Inspiration

The main component of the nervous tissue, that can process information, is the neuron or nerve cell. It is an electrical excitable cell, that can communicate with other neurons over synapses.

A typical Neuron consists of three main components, a cell body, dendrites and an axon, see 2.1. The dendrites are responsible for receiving signals from other neurons in form of chemical neurotransmitters. These signals are generally called spikes or action potentials and are collected in the cell body in form of an electrical charge, which is called the membrane potential. Spikes can either be excitatory or inhibitory.

Excitatory spikes increase the membrane potential and inhibitory spikes restrain the receiving neuron making it less likely for it to spike. If the membrane potential surpasses a certain threshold (typically -50 mV) a new spike is generated in the cell body at the axon hillock and will travel down the axon. At the end of the axon is the presynaptic terminal, which converts the spike into chemical neurotransmitters, which are then again received by the next neuron. The gap between the presynaptic terminal and dendrites, through which the neurotransmitters travel, is called synapse.

After a spike is emitted the membrane potential will return to a resting state, which is typically -70 mV. The membrane of the neuron is not perfect, so over time the neuron will slowly automatically return to its resting state [20a; 17a; Wnu]. The neuron which emits the spike is generally called the presynaptic neuron, whereas the receiving one is called the postsynaptic neuron.

2.2 Artificial Neural Network History

The first electrical model of a Neuron was presented by W. McCulloch et al. [MP43] in 1943. The model has multiple inputs $x_1, ..., x_n$ which are either 0 or 1. These inputs are weighted by manually set weights $w_1, ..., w_n$, that can be negative for inhibitory connections and positive for excitatory connections. The inputs are then summed up and if the sum is greater than a threshold t the output $f(x)$ will be 1 and otherwise 0.

$$g(x) = w_1 * x_1 + ... + w_n * x_n = \sum_{i=1}^{n} w_i * x_i, \quad f(x) = \begin{cases} 1, & \text{if } g(x) > t \\ 0, & \text{otherwise,} \end{cases} \tag{2.1}$$

In 1958 F. Rosenblatt [Ros58] worked on understanding the decision process in the eye of a fly. Doing so he proposed the idea of a perceptron, which he called the Mark I Perceptron. It was modelled after the Neuron by W. McCulloch et al., but had learnable weights, which were set by minimizing the distance between the desired and actual output.

In 1960 ADALINE, which stands for ADAptive LINEar 2.2, was developed by B. Widrow et al. [WH60]. Their neuron model is very similar to the one of F. Rosenblatt but would output -1 if the threshold is not surpassed and it has an extra weighted input, that sets the threshold and is permanently connected to 1.

After the presentation of ADALINE, the "Golden Age" of AI started, people used to believe that everything could be solved with the help of Neural Networks. But this had an abrupt stop in 1969 when M. Minsky et al. [MP69] presented a paper in which

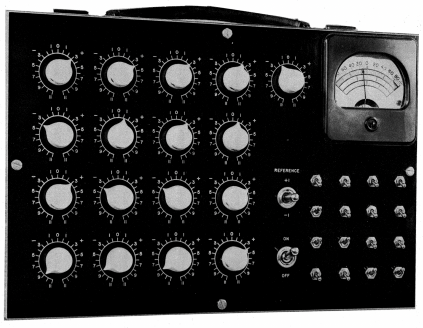


Figure 2.2: Picture of the physical ADALINE computer. The knobs on the left correspond to the weights for the inputs and the toggle switches to the inputs which can either be off or on, so 0 or 1 respectively. Picture is taken from [WH60].

they proved, that perceptrons were only able to solve liner problems and would fail on problems like XOR.

The period afterwards is known as the "AI Winter" and lasted until 1986 when D. Rumelhart et al. [RHW86] presented the Backpropagation algorithm for Multi-layered Perceptrons, which is used until today. The algorithm repeatedly updates weights in the network using Stochastic Gradient Descent and partial derivatives to minimize the distance between the desired and actual output. The use of partial derivatives makes it possible to use this algorithm independent of the network structure.

In 1989 Convolutional Layers were introduced [LeC+89], which are one of the main component of today's deep learning architectures, that use images as input.

The next big step was the Support Vector Machine (SVM) by V. Vapnik et al. [CV95], which is based on statistical learning. The algorithm finds a nonlinear boundary in a dataset consisting of two classes, that maximizes the margin between the two classes.

2.3 Spiking Neural Networks - SNNs

Spiking Neural Networks are considered to be the third generation of Neural Networks and were introduced by W. Maass [Maa97] in 1997. They implement the behavior of biological neurons more closely, than second generation artificial neurons.

One of the most popular groups of neuron models is called Integrate and Fire Models.

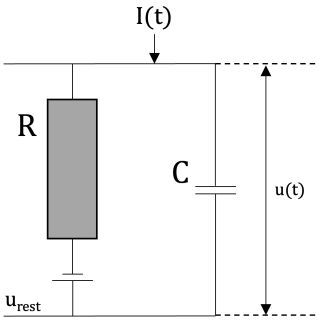


Figure 2.3: Electrical circuit for a Leaky Integrate and Fire Neuron. R is the resistor and C the capacitor.

The neural dynamics of these models can be conceived as a summation process of input spikes, together with a mechanism that triggers spikes above a certain threshold value.

To build a phenomenological model of the neuronal dynamics, a threshold ϑ and a function that describes the voltage potential $v(t)$ within the neuron are needed. The threshold needs to be passed from below for the neuron to trigger a spike. Every spike has the same shape, so that a spike itself does not contain any information, only the time of the spike. Because of that spikes are considered to be events. Neurons are connected with weighted synapses, where the weight translates to the voltage with which the postsynaptic neuron will be charged in the case of a presynaptic spike. This weight can be negative for inhibitory connections or positive for excitatory connections.

The most popular model of this group, which is also used in this thesis, is called Leaky-Integrate and Fire (LIF) Neuron. The equation for it links the momentary voltage $v(t) - v_{rest}$ to the input current $I(t)$, where $v(t)$ can be seen as the membrane potential and u_{rest} as the resting state.

If a current pulse $I(t)$ arrives at the membrane of a biological neuron the additional electric charge $q = \int I(t')dt'$ will charge the cell membrane. Therefore, the membrane acts like a capacitor C and as it is not perfect the charge will leak over time, which can be characterized by a finite leak resistance R.

This behavior can be implemented by a simple electrical circuit with a capacitor C in parallel with a resistor R driven by a current $I(t)$, see 2.3.

The driving current $I(t)$ can be split into two components,

$$I(t) = I_R + I_C$$

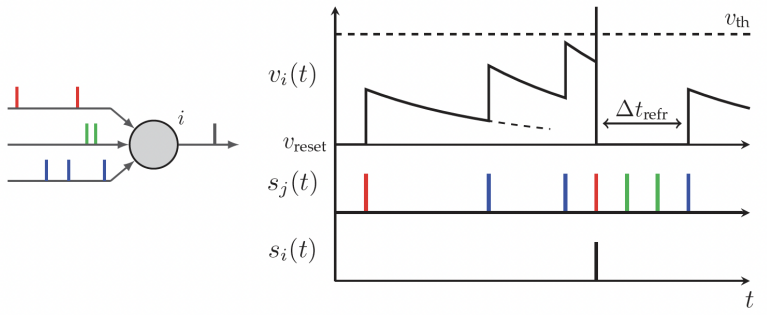


Figure 2.4: Figure displays the temporal course of the membrane potential $v(t)$ of a neuron i. The potential is driven by multiple arriving spikes over 3 synapses, which are depicted as vertical bars. After the threshold v_{th} is surpassed the membrane potential goes back to its resting state v_{rest}. Plot is taken from [PSD19].

The resistive current I_R, which passes through the linear resistor R, can be calculated from Ohm's law, $I_R = (v(t) - v_{rest})/R$ and I_C charges the capacitor C. From the definition of the capacity $C = q/v(t)$, where q is the charge and $v(t)$ the voltage, the capacitive current can be constructed $I_C = dq/dt = C\,dv/dt$. Thus

$$\begin{aligned} I(t) &= \frac{v(t) - v_{rest}}{R} + C\frac{dv}{dt} \\ \Leftrightarrow \tau_m \frac{dv}{dt} &= -(v(t) - v_{rest}) + RI(t), \end{aligned} \tag{2.2}$$

where $\tau_m = RC$ is the membrane time constant of the neuron.

If one waits long enough, the membrane potential $v_{rest} + \Delta u$ is expected to decrease over time until it reaches its resting state v_{rest}, if no spikes arrive in between. Looking at the solution of the differential equation with initial $v(t_0) = v_{rest} + \Delta v$ and no input $I(t)$

$$v(t) - v_{rest} = \Delta v\, e^{-\frac{t-t_0}{\tau_m}} \; for\; t > t_0,$$

one can see that this is the exact behavior. A typical value for the membrane time constant τ_m, which is also called the characteristic time of the decay, is 10ms [Ger+14].

After the neuron fires a spike its voltage potential $v(t)$ will fall back to its resting state v_{rest}. A refractory period can be set t_{refr} in which the voltage potential is not increased by an arriving spike after the neuron spiked 2.4.

2.4 Spike Encoding Techniques

Research of the biological neural systems suggests that most sensory organs encode information into the exact time of a spike rather than using rate encoding techniques.

The human visual system for example needs about 150 ms for object recognition, which supports the theory, as rate encoding techniques would take too much time [TFM96].

This has also been supported by further research, for example about the visual system [GKR96; GM08; Mon+08; Por+16] and audio system [GD43].

For SNNs two main encoding techniques exist. The first one being Rate Encoding and the second one Temporal Encoding.

Rate Encoding techniques use the instantaneous or average rate of spikes of a single or a group of neurons. It can be further categorized into Count Rate, which is the most common rate encoding technique. It is defined as the mean firing rate r over a specific time window, $r = n/t$, where n is the number of spikes in the defined time window t, see 2.5.

Density Rate encoding measures the average spike rate over several runs and Population Rate encoding measures the average spike rate over a group of neurons.

Numerous Temporal Encoding techniques exist, which encode information into the exact time of a spike, the time between two spikes or the order with which neurons of a population spike.

One of the simplest techniques is Time To First Spike (TTFS), which belongs to the group of global referenced temporal encoding techniques. It encodes information by the time difference Δt between stimulus onset and the first spike of a neuron, see 2.5.

For more details on different spike encoding techniques and applications, the survey by Auge et al. [Aug+21] is recommended.

2.5 Spike Time Dependent Plasticity - STDP

STDP is a temporally asymmetric form of Hebbian learning induced by close temporal correlations between the spikes of pre- and postsynaptic neurons [CD+08]. It is believed that this is one way the brain learns, and it helps during development and refinement of neuronal circuits. This has been shown by neuroscience experiments [Mar+97; BP98].

Presynaptic spikes arriving closely before a postsynaptic spike lead in many synapse types to Long-Term Potentiation (LTP), while presynaptic spikes arriving closely after a

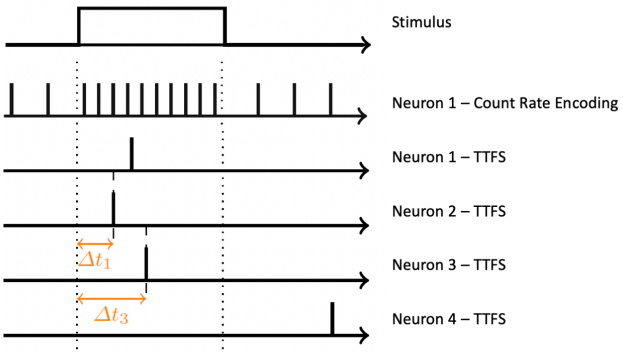


Figure 2.5: Overview of Count Rate Spike encoding and Time To First Spike encoding. The Stimulus shows an input, and the neurons a possible response with the corresponding encoding technique. For the Count Rate Encoding information is encoded in the total number of spikes during the stimulus, not their time. For the TTTFS neurons information is encoded by the exact time difference between start of stimulus and the first spike. Plots are taken from [Aug+21].

postsynaptic spike lead to Long-Term Depression (LTD). LTP and LTD can only happen in a small window around the postsynaptic spike, this window is called the learning window.

The STDP function describes the change of a weight w_{ij} of a synapse from a presynaptic neuron j to a postsynaptic neuron i during the learning window, in correlation to the relative time difference between the time of the presynaptic spike t_j and the time of the postsynaptic spike t_i, see 2.6.

The total weight change Δw_{ij} is then given by

$$\Delta w_{ij} = \sum_{f=1}^{F} \sum_{n=1}^{N} W(t_j^n - t_i^f),$$

where N and F count the total number of pre- and postsynaptic spikes respectively [Ger+96].

$W(x)$ is the learning window, which is often defined as,

$$W(X) = A_+ e^{\frac{-x}{\tau_+}} \quad for\ x > 0,$$
$$W(X) = -A_- e^{\frac{x}{\tau_-}} \quad for\ x < 0.$$

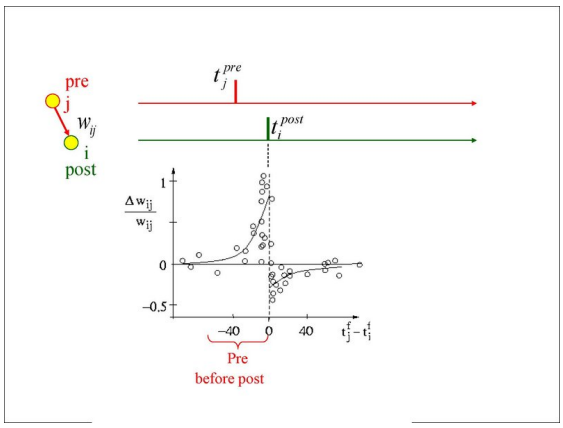


Figure 2.6: Spike Time Dependent Plasticity (STDP)
Figure shows the weight change of a synapses in correlation to the relative timing of pre- and postsynaptic spikes t_j^{pre} and t_i^{post} respectively. Picture is taken from [Ger17].

This is also the chosen definition for this thesis. τ_+ and τ_- are positive time constants that define the width of the positive and negative learning window respectively. Typically, they are chosen to be around 10ms. A_+ and A_- are positive constants scaling the strength of potentiation and depression [SMA00].

2.6 Reward Modulated Spike Time Dependent Plasticity - R-STDP

A learning rule combined with STDP and a reward signal is called R-STDP. The weight change of a synapses is induced by the reward signal, which is distributed in a global manner and can even influence the synaptic weight seconds after a postsynaptic spike, which is due to the synaptic eligibility trace. It describes how long learning after a postsynaptic spike can take place.

The state of a synapses is described by two phenomenological variables the synaptic weight or strength w and a relatively slow processed enzyme acting as the synaptic eligibility trace c.

The synaptic eligibility trace and weight change can be described as

$$\dot{c} = -\frac{c}{\tau_c} + STDP(\tau)\delta(t - t_{pre/post}),$$

$$\dot{w} = cd.$$

Here d describes the extracellular concentration of a neuromodulator dopamine DA, which is the reward. $\delta(t)$ is the Dirac delta function that step-increases the eligibility trace. Spikes of pre- and postsynaptic neurons, occurring at times $t_{pre/post}$ respectively, change c by the amount $STDP(t_{post} - t_{pre})$. The decay rate τ_c controls how long a delayed reward can affect the synapses, which has been shown in experiments to be typically around 5 seconds [OL96].

The concentration of extracellular DA is the same for every synapse and can be modelled as

$$\dot{d} = -\frac{d}{\tau_d} + DA(t),$$

where τ_d is the time constant that determines how long the DA will last. $DA(t)$ models the source of DA due to the activity of dopaminergic neurons [Izh07].

3 Related Work

In general, three steps are necessary for a person following robot. First an algorithm that detects persons needs to be implemented, second that person needs to be tracked and third, the correct commands for the robot need to be computed to follow the person.

Computing the commands for the robot is the main purpose of this thesis, as the tracking information is already provided by the CANN. In literature this last step is quite often not very well described or only superficially described [Cha+20; KMM20; KM16; HM03].

3.1 PID-Controller Based Following Techniques

A common approach to control the driving of the robot is to us PID-Controllers or parts of it.

3.1.1 PID-Controller

A PID controller consists of a proportional, integral and derivative element and they are widely used in feedback control. It combines three different functionalities,

- proportional element (P): represents at any instance in time t the current proportional error
- integral element (I): accumulates past errors and integrates them over time until a time t
- derivative element (D): prediction of the future error, based on its current rate of change

The control function is then given as,

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}.$$

K_p, K_i and K_d are all non-negative coefficients for the respective parts. If the application only needs one or two parts of the PID-Controller, the respective coefficients can be set to 0. Depending on which coefficients are set to the 0, the resulting controller is often named only by the parts which are unequal to 0. For example, PI-Controller if K_d is 0.

In the literature the equation is often formulated as

$$u(t) = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right),$$

where K_i is replaced by K_p/T_i and K_d is replaced by $K_p T_d$. This has the advantage that T_i represents the integration time and T_d the derivative time. $K_p T_d$ is then the time constant with which the controller will attempt to approach the set point. K_p/T_i determines how long the controller will tolerate the output being consistently unequal to the desired output [Paz01].

3.1.2 PID-Controller Approaches

One of the first approaches for a person following robot, was published by Sidenbladh et al. [SKC99] in 1999. Their system receives images from a monocular camera, which are converted into the HSV space. From here large areas with skin are detected and the largest one is considered to be the head of a person, which is then tracked by Taylor approximations.

These visual measurements are used to calculate the error E of the target to a goal position, which is the center of the image (x_0, y_0), $E = [x - x_0, y - y_0]$.

From this a pan angle can be computed to recenter the person on the x axis. The algorithm uses a P-controller for this, but no further description on how a constant distance or how the wheel velocities are calculated is given.

Gockley et al. [GFS07] presented another approach using a laser to recognize the person and track it over time. To keep a constant distance to the target, their system uses a proportional feedback loop over time based on the current distance of the robot to the desired distance. In addition, the error is integrated over time to reduce oscillation behavior. Even though it is not explicitly mentioned in the paper, this is the behavior of a PI-Controller. They do not give more details about the driving controller for the robot.

In 2017 Chen et al. [CST17] presented a person following robot, that used a CNN to detect and track a person. The output of the network is the centroid and distance to the target person.

Their system uses for both the angular velocity ω and linear velocity v PI-controllers. The velocity v error is directly proportional to the distance error between the desired distance to the person and the actual distance to the person. The angular velocity ω error is proportional to the difference $x - x_{mid}$, where x is the x position of the target in the image and x_{mid} the center of the image.

Another approach, that uses multiple CNNs to detect and track a person, was presented in 2018 by Condés et al. [CC18]. Like the previous approach they also use two PID-controllers to calculate the angular velocity ω and linear velocity v, but they include the differential part. The PID-controllers also receive the angular and distance error, that are calculated in the same way as in the previous approach.

3.2 ANN Based Following Techniques

With the unstoppable rise of ANNs in recent years, they also have been applied to the problem of person following robots.

In the paper by Dewa et al. [DM21] multiple agents were trained in different environments by using deep reinforcement learning. Optimal policies are subsequently selected and used for training in following scenarios.

Their network predicts the linear velocity and angular velocity for the robot to follow the person, which are combined with the previous prediction to smooth the driving behavior.

Pang et al. [Pan+20] presented a hybrid approach in which they first train a CNN, with fully connected layers in the end, in a supervised manner and then with reinforcement learning. The network uses a monocular camera as input and outputs a finite set of actions for the robot to follow the person.

3.3 Other Following Techniques

A driving controller approach, which has been used more often in following publications, was presented by Satake et al. [SM09] in 2009. They use a stereo camera as input from which they extract a depth map and depth templates are used to find possible targets within the map. In case they found a possible match a Support Vector Machine is used to verify, that the found person is a real person. This is a very popular detection technique, which can be found in other person following scenarios [HM03; DYC17;

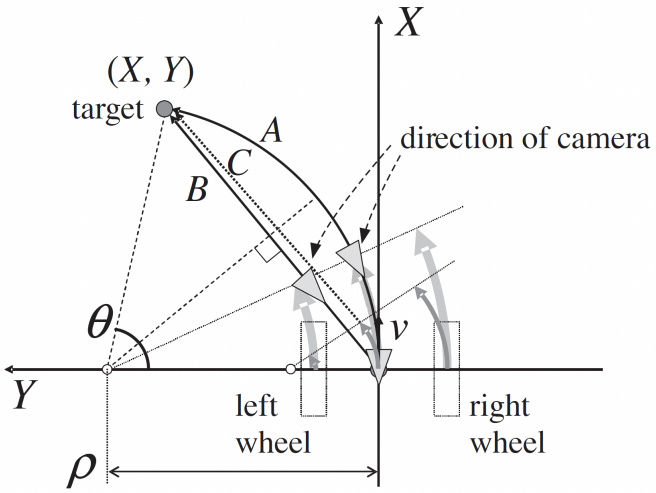


Figure 3.1: Three different paths to the target Position (X, Y) were considered. In B the robot turns first and drives on a straight line towards the goal. Path A has a continuous curvature, which is not optimal as the robot has a slow turning speed. Path C was chosen, which has a reduced turning radius. Figure taken from [SM09].

Cha+20]. For tracking an Extended Kalman Filter (EKF) is used. The tracking and detection system has been improved two times in following years [Miu+10; SCM12].

The driving controller receives a target position as (X, Y) in the robot's coordinate system. It then calculates a circular trajectory to reach this position. For that the velocity of the left and right wheels are calculated by

$$v_L = v\left(1 - \frac{d}{\rho}\right) = v\left(1 - \frac{2dY}{X^2 + Y^2}\right),$$
$$v_R = v\left(1 + \frac{d}{\rho}\right) = v\left(1 + \frac{2dY}{X^2 + Y^2}\right).$$

v is the estimated velocity of the person, which is calculated by the distance the person moved in the robots' coordinate frame and the robot's own velocity. $2d$ represents the distance between the robot's wheels and $\rho = (X^2 + Y^2)/2Y$ describes the turning radius of the robot.

This equation would lead to a continuous circular path, see 3.1 path A. But the robot's turning rate is relatively low, so that there is a high chance of loosing the person. To solve this problem, the turning radius is reduced by ρ/k,

$$v_L = v\left(1 - k\frac{2dY}{X^2 + Y^2}\right),$$
$$v_R = v\left(1 + k\frac{2dY}{X^2 + Y^2}\right).$$

See 3.1 path C.

Jia et al. [Jia+13] used a Stereo camera and a RFID chip to detect and track people. They then use the exact same approach to extract the goal position for the robot from the visual input data. Computing the commands for the velocity of the left and right wheels is also done in the same manner, they only update k online depending on the persons velocity and location with a set of fuzzy rules.

The same system to extract a goal location from visual input data was used here [Ren+16], but they do not go into detail about the wheel velocity calculations.

Other approaches use existing solutions to calculate the driving commands for the robot, like the robuLAB10 [Doi+12] platform or the ros move_base package [DYC17], which are not further explained in the papers.

Here [Kau+19] a manually constructed decision tree that uses the persons 3D position, in the robot's coordinate system, as input data, is used to compute the driving commands.

3.4 Spiking Neural Networks Based Following Approaches

Some approaches can be found that use Spiking Neural Networks. Even though they do not follow persons, but lines or walls, the general idea is quite similar.

Wang et al. [Wan+09] build a wall following robot in 2009. Their system has eight input neurons which either have distinct functionalities, for example to spike if the robot is too close to the wall, or they receive data from one of the sonar sensors. The Neurons which are not connected to a sensor, are connected by synapses to a hidden layer consisting of three neurons. The output layer consists of two neurons which are responsible for predicting the left and right wheel velocities. Weights for the synapses were set by hand.

To calculate the angular ω and linear velocity v of the robot, the following equation is used:

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{2b} & -\frac{r}{2b} \end{pmatrix} \times \begin{pmatrix} n_1 * p \\ n_2 * p \end{pmatrix},$$

where $n_{1/2}$ are the number of output spikes of the respective velocity neuron and p the velocity per spike. r is the radius of the wheels and $2b$ the distance between the wheels.

Kaiser et al. [Kai+16] build a lane following robot in 2016 with SNNs. They used a DVS camera as input, from which pixels are aggregated into groups, where each group has a spike generator. The spikes generators are connected to two sensory neurons, which are connected in a one-to-one fashion to two output neurons. The output neurons indirectly predict the left and right wheel velocities, from which the linear and angular velocities for the robot can be computed. Weights within the network were set manually.

The output velocities are computed by counting the number of spikes n_t in the simulation time step and dividing it by the maximum number of spikes in the time step n_{max}

$$m_t^{left/right} = \frac{n_t}{n_{max}}.$$

The turn velocity is defined as,

$$S_t = c_{turn} * (m_t^{left} - m_t^{right}),$$

where c_{turn} is a turn constant. To reduce the turn velocities, the overall velocity is controlled by,

$$V_t = -|m_t^{left} - m_t^{right}| * (v_{max} - v_{min}) + v_{max},$$

where $v_{max/min}$ are velocity boundaries of the robot.

The linear velocity and turn velocity are the weighted sum between the previous and current velocity,

$$v_t = c * V_t + (1 - c) * v_{t-1}, \quad s_t = c * S_t + (1 - c) * s_{t-1}.$$

c depends on the spike amount of the left and right velocity neuron

$$c = \sqrt{\frac{m_{t-left}^2 + m_{t-right}^2}{2}}.$$

The linear and angular velocity are then used to steer the robot.

Bing et al. [Bin+18] implemented a line following robot in 2018 using a SNN with two layers, that has been trained with R-STDP. The network receives its input from a dynamic vision sensor (DVS). Multiple pixels are grouped together and multiple of these groups are connected to one neuron of an 8x4 grid of neurons. The output layer

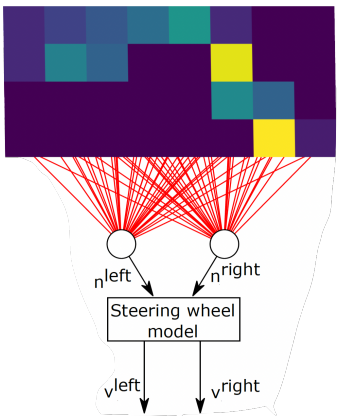


Figure 3.2: Architecture of [Bin+18] implementation of a line following robot. The first layer is a grid of 8x4 neurons which are connected in an all-to-all manner to the second layer of output neurons. The first layer receives its input from a DVS. Figure taken from [Bin+18].

consists of two output neurons, which are responsible for indirectly predicting the left and right wheel velocities of the robot. The first and second layer are connected in an all-to-all manner 3.2.

To calculate the angular and linear velocity the same technique is used as in the paper by Kaiser et al. [Kai+16], with the difference, that the robot receives velocities for the left and right wheel, which are calculated as

$$v_t^{left} = v_t + s_t; \quad v_t^{right} = v_t - s_t.$$

On year later in 2019 Bing et al. [Bin+19] proposed another approach for a target reaching robot, that avoids obstacles. Their focus was to create a SNN that could be used for a variety of problems and was not only designed for a specific task as it is often the case with SNNs. The SNN consists of 3 Layers, an input layer, a hidden layer and an output layer and can be trained using R-STDP. The overall structure consists of one SNN that receives sensor data and is responsible for target reaching. The output is then fed in parallel to two other SNNs, where the first one is responsible for obstacle avoidance (OA) and the second one for goal approaching (GA). They both output turn angles for the robot to turn left or right $\alpha_{\mathrm{OA/GA,\,L/R}}$.

A set of rules is defined, depending on the output of both SNNs, to choose which angle to use for the calculation of the final angle. The final turn angle α for the robot is then calculated as

$$\alpha = \alpha_{min,\mathrm{OA/GA}} + (\alpha_{max,\mathrm{OA/GA}} - \alpha_{min,\mathrm{OA/GA}}) \times (y_{SNN}/y_{max}),$$

where y_{SNN} and y_{max} are the spike output and the maximum spike output of the SNN and α_{min} and α_{max} are the range of the turning angle.

The final velocities for both wheels are then calculated as

$$v_{left} = v_{forward} - \Delta v(\alpha)/2,$$
$$v_{left} = v_{right} + \Delta v(\alpha)/2,$$

where $v_{forward}$ is a default velocity given at 5 rad/s and $\Delta v(\alpha)$ the difference in motor velocities necessary to achieve a turn of α degrees in 1 second.

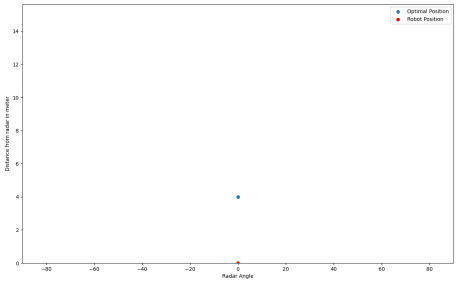


Figure 4.1: Distance to the Person, which the robot has been trained to keep. The robot is positioned at (0,0) and the plot is from the perspective of the radar sensor and is in polar coordinates.

4 Approach

The following section describes the approach taken to construct a person following robot. The focus of this thesis was to construct a network which takes as input the tracking signal of a person and to compute the driving commands for the robot from this.

In the following a step describes a simulation interval which is equal to 100ms. During this interval the network receives the tracking signal in form of spikes and its output spikes are recorded to calculate the velocity of the robot.

The optimal position describes the distance the robot should have from the person. The robot was trained to keep this distance and errors are calculated as the difference to this position. The distance is 4 meters behind the person and the robot should face the person at a 0° angle 4.1.

The network was constructed to work with robots that use differential drive, so it will independently predict the left and right wheel velocities for the robot.

4.1 Input for the SNN

The network receives its input data from a CANN which stands for Continuous Attractor Neural Network. They can be used to encode simple continuous information, such as orientation, moving direction or spatial location of objects [Wu+16]. As person tracking is also a continuous process, a CANN can be applied for this problem, which was done in this master thesis [Aks22]. This CANN was used in this thesis.

The CANN uses FMCW radar data as input, which is in the form of a range angle map. The CANN consists of neurons arranged in a grid, where the x-axis is the angle and the y-axis the radius. The output are multiple spike trains of different neurons, that are arranged in a circle on the grid. Neurons in the center of the circle have the highest spike rate and neurons further from the center have decreasing spike rates until they do not spike anymore. The person is then most likely located in the center of the spike circle 4.2.

Because the input of the radar is continuous, neurons in the CANN always represent an area from the input.

For training and testing the CANN network was simulated in this thesis, with a 2D Gaussian distribution. The center of the Gaussian distribution is at the position of a simulated person and probabilities where set to represent spike rates most closely to the output of the real CANN [Aks22].

The original CANN uses radar data which covers a range of 50 meters and from -90° to 90° and it consists of 64x64 neurons. This means, that each neuron covers a range of 0.78126 meters and 3°. Because of this accuracy of the input data is limited especially in the radius domain. The covered radius from the radar was limited to 15.625 meters in the simulated CANN, so that each neuron covers a smaller radius.

4.2 SNN Architecture

Figure 4.3 shows the architecture which was chosen for this thesis. It was inspired by the paper of Bing et al. who build a lane keeping robot using SNNs [Bin+18]. Every neuron is a Leaky Integrate and Fire neuron, with a membrane time constant τ of 20 ms, a threshold voltage of 100 mV, a reset voltage of 0 mV and no refractory time.

The network receives its input from the CANN which is simulated by a spike generator group. In this case it is assumed, that the CANN has a size of 40 by 40 Neurons. This results in every neuron covering a radius of 0.390625 meters with an angle of 4.5°.

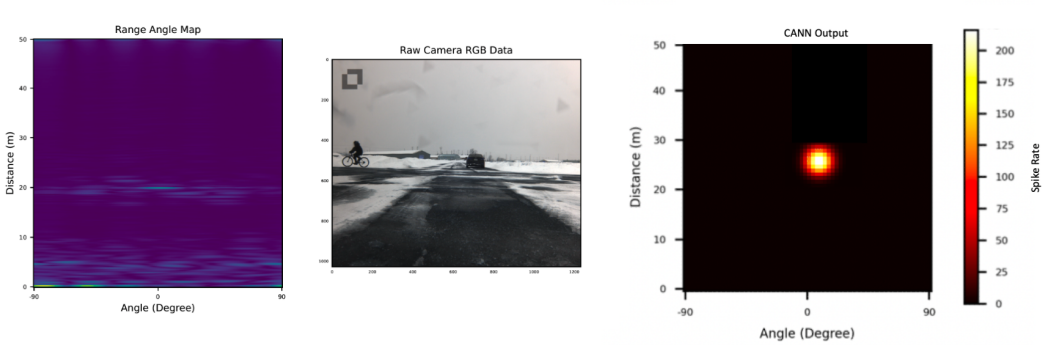


Figure 4.2: The left image shows the synchronized range angle map for the image in the middle, from the CARRADA data set [Oua+21]. The left image shows what the output of a radar looks like [Aks22]. The image on the right shows the CANN output, where the yellow/orange/red area is the spike circle. Notice, that these pictures are taken from this thesis [Aks22]. The range of the y-axis was changed for this thesis.

The simulated CANN then contains 1600 neurons instead of 4096, due to which the simulation is a lot faster 4.1. All these Neurons are connected in an all-to-all fashion to two output neurons with R-STDP synapses. The parameters for the synapses can be found here 4.1.

The two output neurons are responsible for predicting velocities which are added to the left and right wheel velocities.

The weights for synapses are trained using R-STDP 4.6 and the input and output encoding and decoding techniques are described here 4.3.

The problem with this architecture is, that if the network is trained, the weights and therefore predicted velocities are fixed. If the person is in front of the optimal position, so further than 4 meters away, the robot will always accelerate, to keep up with the

CANN to velocity R-STDP Synapses Parameters		
Parameter	Explanation	Value
τ_c	Time constant of eligibility trace	500 ms
τ_r	time constant of dopaminergic trace	200 ms
τ_+	STDP time constant for potentiation	20 mV
τ_-	STDP time constant for depression	20 mV
A_+	Amplitude of weight change for potentiation	0.001
A_-	Amplitude of weight change for depression	0.001
w_{min}	Minimum weight of synapses	-200 mV
w_{max}	Maximum weight of synapses	+200 mV
c_{min}	Minimum weight of synaptic eligibility trace	0 mV
c_{min}	Maximum weight of synaptic eligibility trace	100 mV

Table 4.1: Values for the R-STDP Synapses connecting the simulated CANN to the velocity prediction neurons.

person. On the other hand, if the robot is too close to the person, the robot will always decelerate to have a greater distance to the person. Because of that, if the person is in front of the optimal position, the robot will accelerate even if the person gets closer to the optimal position. If the person surpasses the optimal position, the robot's velocity will be much greater than the persons velocity and the robot will start to decelerate. The same happens now when the robot decelerates, only the robot's velocity will be much smaller than the persons velocity when the person surpasses the optimal position. This leads to a highly oscillating behavior and to prevent this, the distance measuring network was added.

4.2.1 Distance Measuring Network

The distance network measures if the person gets closer or further away from the robot. With that information the robot can be decelerated or accelerated, depending on if the robot needs to be faster to not surpass the optimal position or slower.

This behavior is achieved by the architecture in 4.4. Every neuron in the CANN is connected to one neuron (continuous circle) in the distance network, that represents the current position at t. From here on out these neurons will be called t neurons. Each of these neurons is mirrored by a second neuron (dotted circle), which "remembers" the last position of the person $t-1$. From here on out these neurons will be called $t-1$ neurons. If a t neuron spikes, the spike will also be sent to its corresponding $t-1$ neuron. This neuron will spike delayed in the next simulation step at the exact same

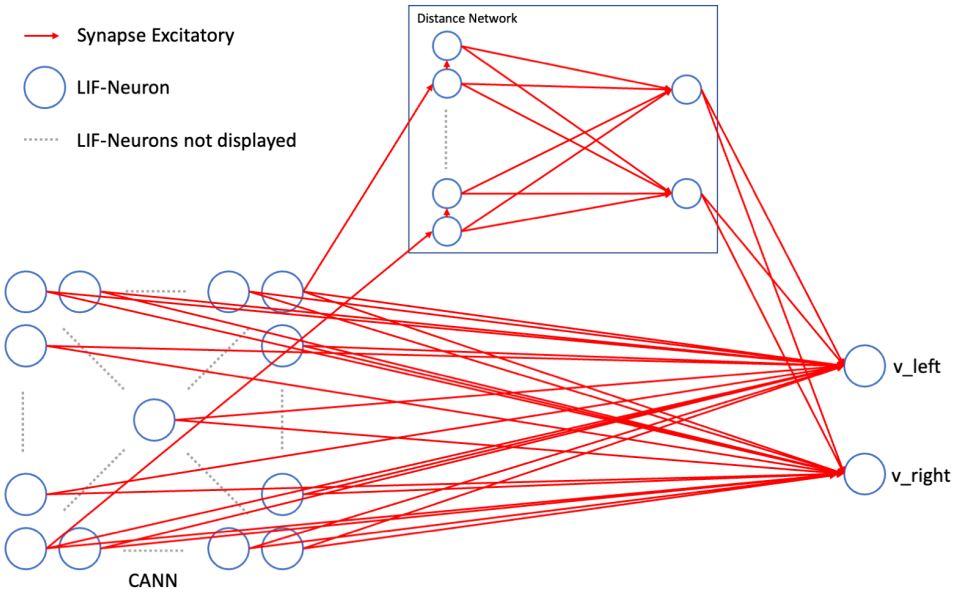


Figure 4.3: Network Architecture - The input comes from the CANN, which is connected in an all-to-all fashion to two output neurons that predict the velocity. The distance network measures if the person gets closer or further away from the optimal position and influences the predicted velocity.

time, with respect to the time in the time step. Each of these neurons will be connected to two output neurons, one of which will spike if the person comes closer to the robot and the other one if the person gets further away.

This is achieved by the weights chosen for the synapses, which have not been trained but set by hand, which was more efficient in this case. The exact value of the weight is not important, they only need to steadily increase the further the corresponding neuron in the CANN is away from the robot's position. This means, that neurons corresponding to the same radius have the same weights. The t neurons are connected with each other with inhibitory synapses to guarantee, that if one of them spikes, no other will spike in the same step. Because the neuron in the center of the neuron circle of the CANN will spike first, also the corresponding neuron in the distance network will spike first and no other will spike afterwards in that step.

All t neurons are connected with excitatory synapses to the neuron which will spike if the robot is further away and with inhibitory synapses to the neuron which will spike

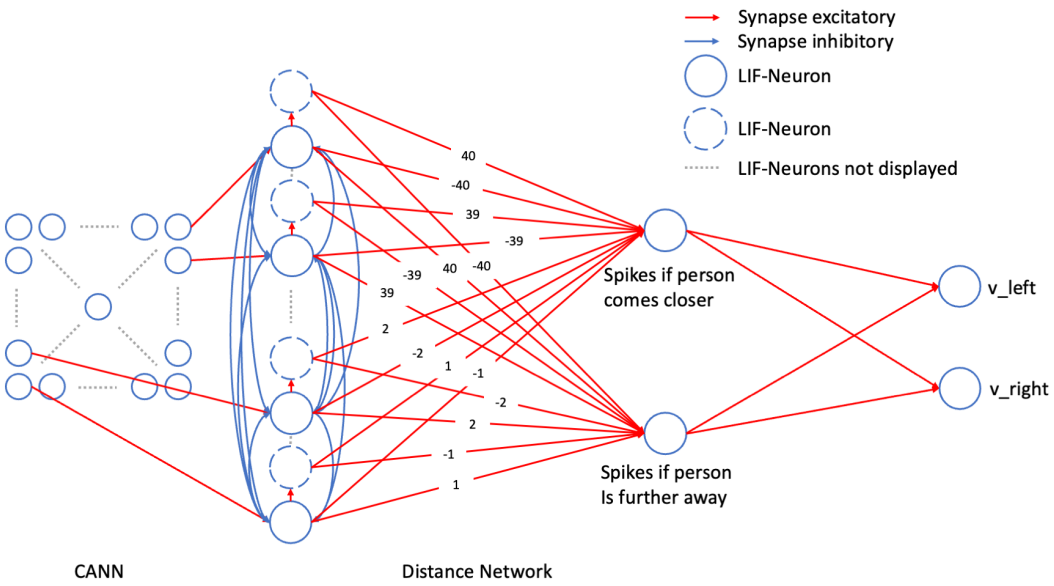


Figure 4.4: Distance Network Architecture - The distance network measures if the person gets closer or further away from the optimal position and influences the predicted velocity. The t neurons represent the current position of the person, whereas the $t-1$ neurons represent the previous position of the person. One of the two output neurons will spike if the person gets closer and the other one if the person walks further away. This behavior is achieved by the combination of inhibitory and excitatory synapses.

if the robot is closer. All $t-1$ neurons are connected the other way around.

Now, due to the increasing absolute values of weights of synapses, only one of the two neurons for the distance measure will spike. If the robot moves away, the weight of the excitatory synapse that connects the t neuron with the "robot further away" neuron will be greater than the negative weight of the inhibitory synapse of the $t-1$ neuron connecting it to the "robot further away" neuron, which will then spike. As the synapses weight for t neuron to the "robot closer" neuron has a greater negative weight, than the positive weight of the $t-1$ neuron, the "robot closer" neuron will not spike.

This behavior is exactly flipped if the person comes closer.

The "robot further away" and "robot closer" neurons are connected in an all-to-all fashion to the velocity neurons and by assigning much greater weights to them then

Distance to velocity R-STDP Synapses		
Parameter	Explanation	Value
τ_c	Time constant of eligibility trace	105 ms
τ_r	time constant of dopaminergic trace	105 ms
τ_+	STDP time constant for potentiation	20 mV
τ_-	STDP time constant for depression	20 mV
A_+	Amplitude of weight change for potentiation	0.001
A_-	Amplitude of weight change for depression	0.001
w_{min}	Minimum weight of synapses	-300 mV
w_{max}	Maximum weight of synapses	+300 mV
c_{min}	Minimum weight of synaptic eligibility trace	0 mV
c_{max}	Maximum weight of synaptic eligibility trace	100 mV

Table 4.2: Values for the R-STDP Synapses connecting the distance measuring network to the velocity prediction neurons.

the other synapses connected to the velocity neurons, their output spiking behavior can be influenced, see 4.3.

The weights have also been trained using R-STDP and the parameters for the synapses can be found in table 4.2.

The whole network can be viewed as a PD-controller, as the network without the distance measuring network increases the robot's linear velocity and angular velocity proportional to the current error, which represents the P element of the controller.

The distance measuring network anticipates the future error, by measuring if the person comes closer to the optimal position or gets further away. This represents the D element of the controller.

4.3 Encoding/Decoding of Spikes

This section describes what encoding and decoding techniques were used.

4.3.1 Spike Input Encoding

As describe in the section input for the SNN 4.1 the original CANN uses count rate decoding. This has the disadvantage, that the network emits a great number of spikes, which makes it inefficient compared to other spike encoding techniques. Also, the proposed solution against oscillating behavior of the robot would not work anymore.

For these reasons the simulated CANN uses time to first spike encoding, where an early spike corresponds to a high spike rate in the original CANN. To construct the spike times a 2D Gaussian distribution over the Radar area is used, which is centered around the person's position. The covariance matrix for the distribution is given as

$$\begin{pmatrix} 20 & 0 \\ 0 & 20 \end{pmatrix}.$$

Afterwards multiple operations are applied, to be as close as possible to the original CANN. First the probability values are scaled by $\sinh(\exp(p))$, where p is the probability, and mapped between 0 and 1. Then all values under 0.4286 are set to 0, as otherwise every neuron in the simulated CANN would spike eventually. In the last step every probability value greater than 0.619 is squared, while the others are multiplied with 0.8. The spike time can then be computed as

$$t_{spike} = 0.1 * \left(1 - \frac{p}{100}\right); \quad p \in (0, 1],$$

if a simulation step is 100 ms. t_{spike} is given in seconds for the respective simulation step.

4.3.2 Spike Output Decoding

The velocity neurons in the network also use time to first spike encoding. To map the spike time to a velocity change the following equation is used, which is identical for both sides:

$$v_{\text{diff-left/right}} = -\frac{t - 50}{100}; \quad v_{\text{diff-left/right}} \in [-0.5, 0.5]\text{m/s}; \quad t \in [0, 100] \text{ ms}.$$

Here t stands for the time of the first spike in the step and is relative to the step not the total simulation duration. This means, that spikes occurring early will lead to positive predicted velocity changes, spikes occurring exactly at the half of the step will not affect the velocity and spikes occurring towards the end will lead to negative predicted velocity changes. The output value is negated to conform with the reward functions.

As of now, if the output neuron does not spike in a time step, it is treated the same way as if the network would spike at the end of the step, so the velocity change is -0.5 m/s.

The distance network can now move the spike time forward or backward by having comparatively big absolute weights on the synapses connecting it to the velocity neurons. If the person moves away, the weight of the synapses, connecting the spiking

neuron in the distance network to the velocity neurons, moves the spike time further to the front, so that the predicted velocity change will be increased.

If the person comes closer, the negative weight of the synapses, connecting the spiking neuron in the distance network to the velocity neurons, moves the spike time further to the back, so that the predicted velocity change will be decreased.

4.4 Velocity Controller

Before the predicted velocities are added to the current velocities and given to the robot the angular velocity is limited, while keeping the predicted linear velocity. This is achieved by calculating how much the new velocities of the robot need to be adapted, so that the maximum angular velocity is not surpassed

$$a = \begin{cases} -\frac{|v_{\text{left}} - v_{\text{right}}| - d * \omega_{max}}{2}, & v_{\text{left}} < v_{\text{right}} \\ \frac{|v_{\text{left}} - v_{\text{right}}| - d * \omega_{max}}{2}, & \text{else.} \end{cases}$$

Here $v_{\text{left/right}}$ describe the sum of the current left or right wheel velocity and predicted velocity. d is the distance between the two wheels and ω_{max} the maximum angular velocity, for which 0.2 rad/s is used. The final velocities which are given to the robot are calculated as

$$v_{\text{left}} = v_{\text{left}} - a,$$

$$v_{\text{right}} = v_{\text{right}} + a.$$

The velocity for the left and right wheel is also limited between 0 and 2 m/s.

4.5 Reward Function

For training two different Reward functions are needed, one for the synapses that connect the CANN to the velocity neurons and one for the synapses, that connect the distance neurons to the velocity neurons. Reward functions are also often known as loss functions.

Synapses from CANN Neurons to Velocity Neurons

The reward function chosen in this thesis uses the distance to the optimal position.

$$reward_{left} = \alpha * y_{error} + \beta * x_{error},$$

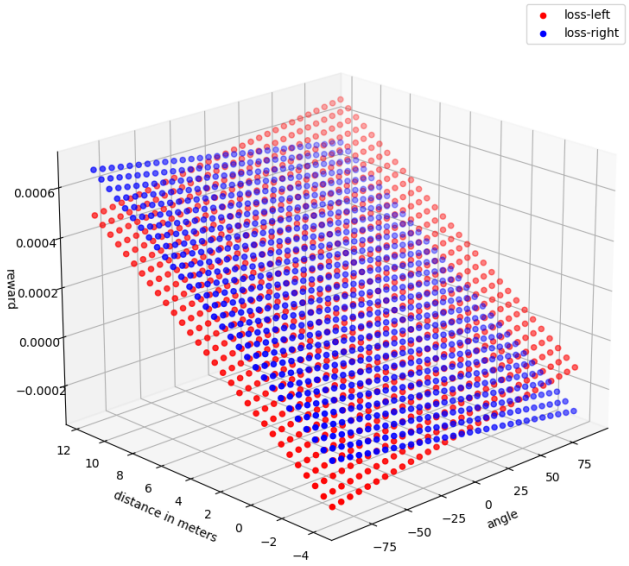


Figure 4.5: Reward function for the synapses connecting the CANN to the velocity neurons. The plot is from the perspective of the optimal position, so robot is positioned at a 0° angle and distance -4 m and values on the distance and angle axis represent positions of the person. Red dots represent the reward for synapses to the neuron predicting the velocity of the left wheel and blue dots the reward for synapses to the neuron predicting the velocity of the right wheel.

$$reward_{right} = \alpha * y_{error} - \beta * x_{error}.$$

The y_{error} describes the radius distance to the optimal position, which is in the range of [-4, 11.625]. The x_{error} describes the angular difference to the optimal position, which is in the range of [-90, 90]. α and β are scaling factors, which are needed to make the influence of both error values more equal, as the x_{error} can be much larger than the y_{error}. Another reason is to scale down the overall error to prevent too big changes in weights of synapses, which would lead to the network not being able to train properly, as optimal weights can easily be stepped over. α was chosen to be 0.00005 and β to be 0.000001 4.5.

Note the difference between the two reward functions. For the left reward the x_{error} is added, while for the right reward the x_{error} is subtracted. This is because, when the person is left of the robot the right wheel needs to turn faster than the left wheel and as the angle error is negative the error needs to be subtracted from the right reward

and added to the left reward, so that the right reward is greater than the left reward. The same happens when the person is on the right, only reversed.

Synapses from Distance Neurons to Velocity Neurons

For the distance reward two factors need to be considered, the robots' velocity compared to the persons velocity and the difference to the optimal position. The velocity only needs to be adapted when the person is close to the optimal position, as this is the point where oscillating behavior can take place, and if the difference between the persons and robots' velocity is great. Also, only the radius error needs to be considered.

$$reward_{distance} = \frac{\gamma * |v_{diff}|}{max(0.1, |y_{error}|)}$$

Here γ is another scaling factor, v_{diff} the difference between the velocity of the person and the robot. Without $max(0.1, |y_{error}|)$ the reward value would go against infinity, if y_{error} would go against 0. Now, as the velocity should only be influenced, if the robot is too slow and the person behind the optimal position (radius < 4 meters to person) or if the robot is too fast and the person is in front of the optimal position (radius > 4 meters to person), three different cases need to be considered

$$reward_{distance} = \begin{cases} reward_{distance}, & \text{if radius < 4 m to person and } v_{diff} <= 0 \\ -reward_{distance}, & \text{if radius > 4 m to person and } v_{diff} > 0 \\ 0, & \text{else.} \end{cases}$$

4.6 Training and Resulting Weights

The training process for the robot is split into different steps with increasing difficulty. During the training, the synapses from the CANN to the velocity neurons and the synapses from the distance network to the velocity neurons are trained in an alternating manner. In table 4.3 the different steps during training are displayed.

To prevent extreme weights in synapses no training was performed if the robot managed to minimize the loss compared to the previous step.

After 1500 steps, the robot's task is to reach the person, which is described in the table as reaching its goal. The robot is considered to have reached the person, if the absolute radius is smaller than 0.3 meters and the absolute angle is smaller than 2°. The simulation is also reset, if the robot needed more than 500 steps to catch up to the

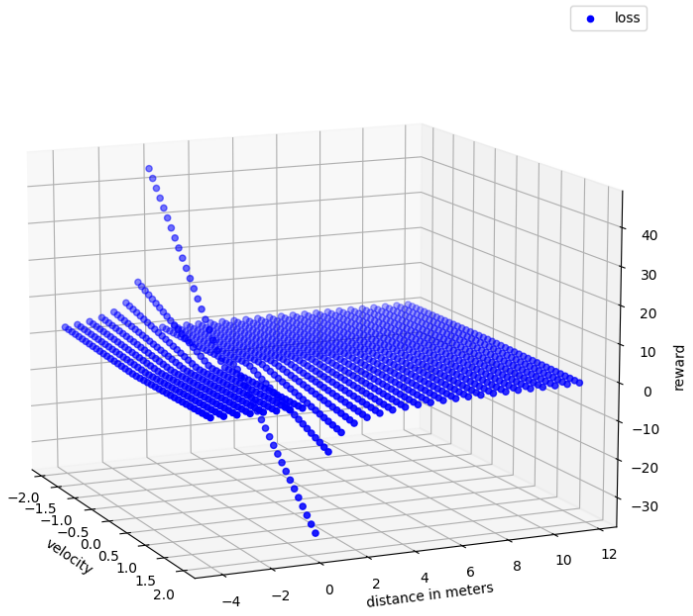


Figure 4.6: Reward function for the synapses connecting the distance network to the velocity neurons. The plot is from the perspective of the optimal position, so robot is positioned at angle 0 and distance -4 and values on the distance and angle axis represent positions of the person. Blue dots represent the reward for all synapses connecting the distance network to the velocity neurons.

person since the last reset. If the simulation would not be reset if the robot reached its goal, only a small part of the synapses would be trained. This is because the robot managed quite early to keep up with the person, so only a small area of the CANN is active.

During training the person walks with a constant velocity of 1 m/s. This can be done because the network does not predict absolute velocities, but only increases or decreases the velocity. This means, that the network does not need to know about the actual velocity of the person.

It was also tested to train the robot while the person changed its velocity, but this led to worse performance. In the case, that the person left the visible are of the robot, the simulation was reset.

Training Procedure	
Steps	Explanation
0-1500	The Person walks in a straight line in front of the robot at a 0° angle and the robot should follow that person
1501-4000	The Person gets placed at a random distance in front of the robot at a 0° angle. If the robot reaches the goal, the simulation is reset.
4001-20000	The person gets placed randomly in the radar area and every 1000 steps the turn probability 4.7.1 for the random walk is increased by 1.2%. If the robot reaches the goal, the simulation is reset.

Table 4.3: Training Procedure for the robot, where the difficulty is increased over time.

4.6.1 Resulting Weights CANN to Velocity Synapses

In figure 4.7 the distribution of weights is displayed. Dark areas represent low weights, whereas bright areas represent high weights. In general, they behave as one would expect. The area where the person has no angular error is very similar for the synapses, as here both wheels should behave similar.

The area left and right of the 0° angle error also look similar to each other, with the difference, that the weights are mirrored.

If the error is smaller than 0° the left wheels weights are lower than the right wheels weights, so that the robot can turn left. If the error is greater than 0° the left wheels weights are bigger than the right wheels weights, so that the robot can turn right.

4.6.2 Resulting Weights Distance to Velocity Synapses

The trained weights for the distance network are very similar to each other. As one would expect the synapses that transfer spike if the person walks away have high weights to increase the velocity of the robot, whereas the synapses that transfer spikes if the person comes closer are negative to slow down the robot.

4.7 Implementation

This section describes how the simulation environment and the network were implemented.

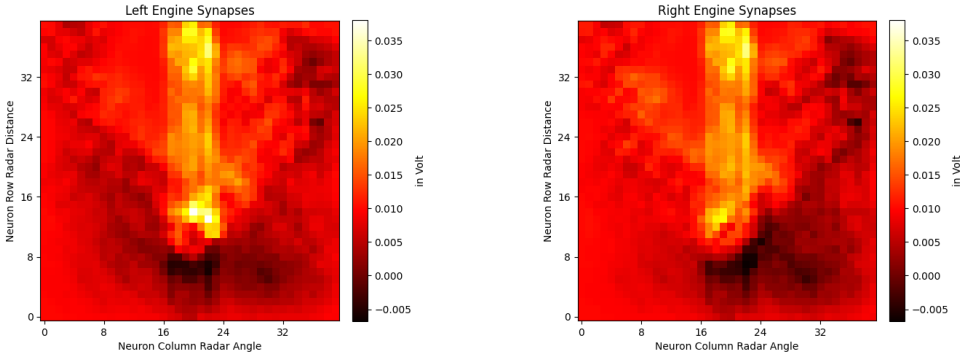


Figure 4.7: Trained weights for the synapses connecting the neurons from the CANN to the velocity neurons.

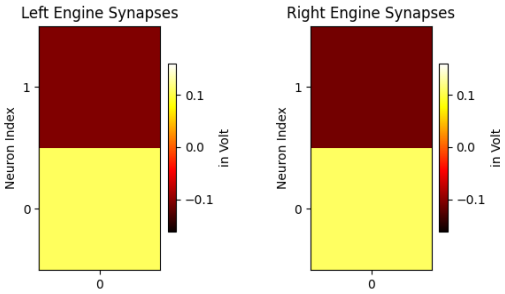


Figure 4.8: Trained weights for the synapses connecting the neurons from the distance network to the velocity neurons.

4.7.1 Simulation Environment

The simulation environment was developed in python 3 and is kept very simple. Its main purpose is to simulate the movement of the person, the robot and the CANN. This data can then be used to compute the reward for the R-STDP synapses and as input for the network.

Using the matplotlib [12b] library, the behavior of the person, robot and CANN have been visualized to judge the behavior of the robot more easily. The visualization contains the path of the person in both cartesian and polar coordinates from the robot's perspective, the simulated CANN and the global plot of the person's and robot's path 4.9.

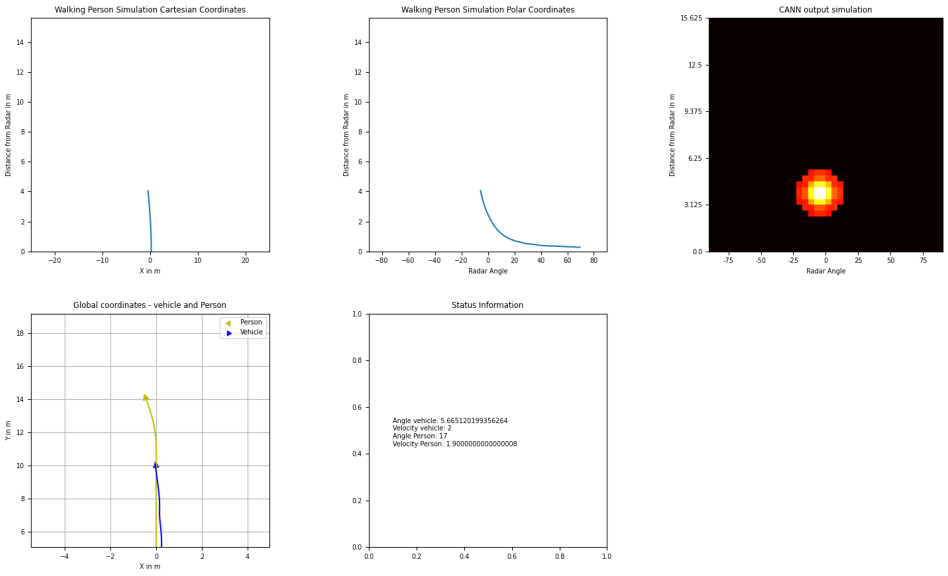


Figure 4.9: Visualization of the Simulation Environment

Person Simulation

To simulate the movement of the person multiple parameters can be defined, which can be changed during the simulation.

In every simulation step the algorithm first checks if the person can change the angle, otherwise the next position is calculated from the current position, velocity of the person and the current angle the person is walking at.

If the person can change the angle, waited minWaitTurnSteps after its last turn and is currently not in a turn, a new goal angle is computed. After that in every step the turnAngleTimeStep is added to the current angle of the person and the new position of the person is computed. This is done until the goal angle is reached and everything starts from the beginning 1.

The new position of the person can be computed as follows:

$$x_{new} = x_{old} + v_p/a * sin(\beta),$$
$$y_{new} = y_{old} + v_p/a * cos(\beta).$$

where v_p is the given persons velocity and a is the number of steps per second, as the velocity is given in m/s and the position for the next step needs to be computed. In this case a step took 100ms, so a was chosen as 10. β is the angle the person is currently

Parameters for the Person Simulation		
Parameter	Explanation	Default
personVelocity	The velocity at which the person is walking.	1 m/s
turnAngleTimeStep	The angle the person will turn in each step.	1°
turnProbability	Probability with which the person will start to turn in each step.	0.2
minWaitTurnSteps	Number of steps the person waits after it turned and walks straight.	20
minTurnAngle	The minimum angle the person can turn before walking straight again.	10
maxTurnAngle	The maximum angle the person can turn before walking straight again.	20
changeAngle	True if the person can change the angle he is walking at	True

Table 4.4: Parameters for the Person Simulation. The values can be changed during the simulation.

walking at. It is assumed, that if the person walks at a 0° angle, it walks in parallel to the y axis.

Robot Simulation

The simulation for the robot is simpler, as the network predicts the change of velocities for the left and right wheel. The algorithm to compute the next position was taken from [17b]. The algorithm assumes, that if the robot drives at a 0° angle it will drive in parallel to the x-axis. To confirm with the person, the algorithm was adapted, so that if the robot drives at 0° angle it will drive in parallel to the y-axis.

First the radius the robot is driving at is computed to get the center of the curvature. Next the angular velocity of the robot is computed and with that the angle change of the simulation step. Having this information, the new position of the robot can be computed.

Algorithm 1 Algorithm to determine the next position of the Person given a set of parameters which are defined in 4.4.

```
cAngle = 0 // the current Angle
gAngle = cAngle // the goal Angle
stepsLastTurn = 0 // number of steps since the Robot finished its last turn
posPerson = (0, 4) // The position the person is currently at
while Simulate do
    if changeAngle then
        randProb = computeRandomProbabilityValue()
        if stepsLastTurn ≥ minWaitTurnSteps and randProb ≤ turnProbability then
            gAngle = determineNewGoalAngle(minTurnAngle, maxTurnAngle)
            stepsLastTurn = 0
        end if
        if gAngle ≠ cAngle then
            if gAngle > cAngle and gAngle > cAngle + turnAngleTimeStep then
                cAngle += turnAngleTimeStep
            else if gAngle < cAngle and gAngle < cAngle - turnAngleTimeStep then:
                cAngle -= turnAngleTimeStep
            else
                cAngle = gAngle
            end if
        else
            stepsSinceLastTurn += 1
        end if
    end if
    posPerson = calculateNewPosPerson(posPerson, cAngle, personVelocity)
end while
```

4.7.2 Network Implementation

The network was implemented in Python 3 [12c] using the Brian 2 Library [12a], which is a free open-source simulator for spiking neural networks written in Python. The library allows the user to define behavior of neurons and synapses to their own need, by providing their own equations.

In this case the equations shown in the background section 2 have been used. For more details on how to setup a network, it is recommended to visit the tutorial section of the brian2 simulator. Here everything is explained in detail, and numerous examples can be found.

Setting up the network itself is straight forward. The only thing which should be pointed out here is how to induce the reward signal.

When looking at the equation for the concentration of extracellular DA,

$$\dot{d} = -\frac{d}{\tau_d} + DA(t),$$

one can see that the $DA(t)$ part needs to be simulated to modify the artificial concentration of extracellular DA. This is done with the help of a new set of synapses, which connect a group of spike generators to every R-STDP synapses in a one-to-one fashion. In every simulation step the spike simulator simulates a spike for each synapse and the reward signal is set as the weight of the synapses which is then added to the extracellular concentration of DA, in the form of $DA(t)$.

This will only change the weight of synapses which received a spike in the duration of the synaptic eligibility trace, as the weight change depends on the multiplication of the eligibility trace and the reward signal. For more details the example by Dumas is recommended [Dum18].

Using the matplotlib [12b] library, the behavior of the network has also been visualized for convenience. The visualization contains the input spikes and output spikes of the current simulation step, the weights of all R-STDP synapses and their corresponding eligibility trace 4.10.

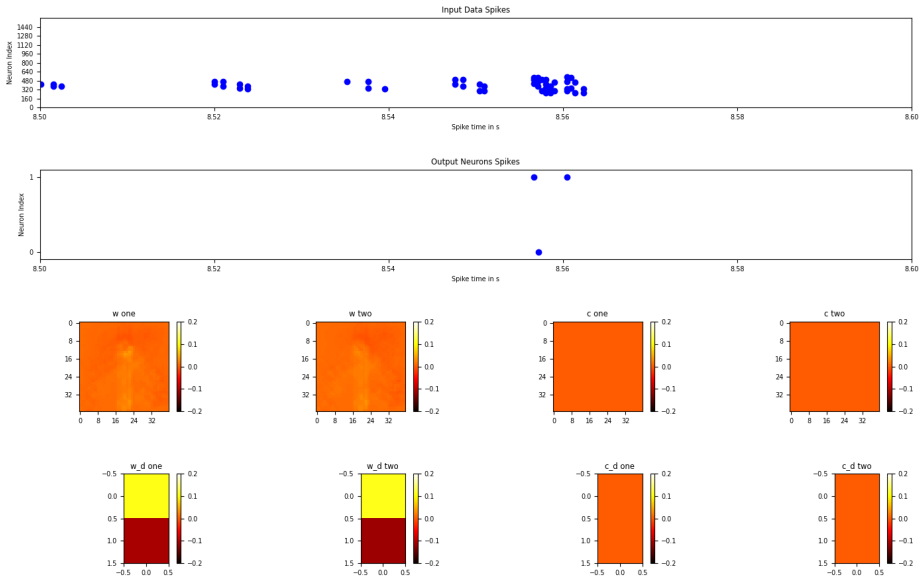


Figure 4.10: Visualization of the Network Environment

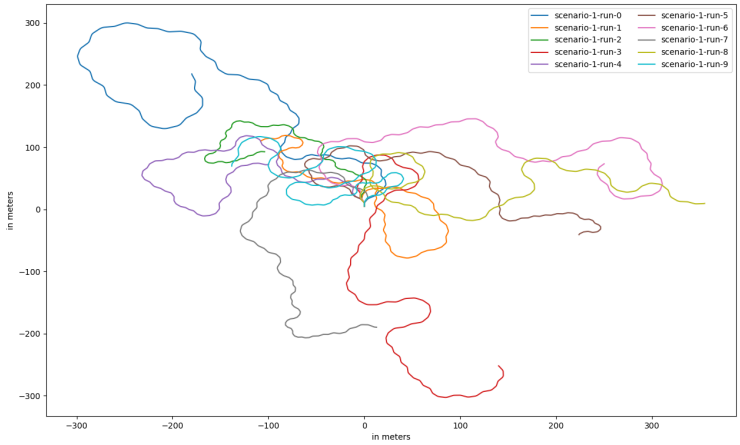


Figure 5.1: Scenario 1 - Different paths walked by the person. Different lengths result from different randomized velocities during the simulations.

5 Evaluation

To evaluate the performance of the proposed architecture five different experiments have been conducted in the same environment in which the robot has been trained.

5.1 Scenario 1 - Random Paths

The first experiment was to let the person start 1 meter and at 0° angle in front of the optimal position. The Person then started to walk a random path, where in each step the velocity got randomly increased or decreased by $\pm 0.01 \mathrm{m/s}$, or stayed constant. Each of these events occurred with the same probability and the velocity stayed in [0, 2] m/s. The total duration of the experiment was 10 minutes or 6000 steps and it was repeated 10 times. In 5.1 one can see the paths which have been walked by the person.

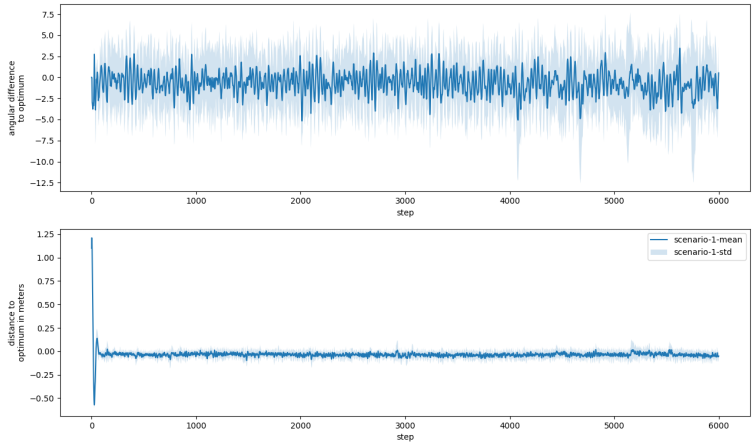


Figure 5.2: Scenario 1 mean error and standard deviation over the 10 runs with random walking. The angular error stays mostly within $\pm 7.5°$ and the distance error within $\pm$10cm.

5.1.1 Scenario 1 - Result

In each of the 10 simulations the robot managed to keep up with person until the end. Figure 5.2 displays the mean error and standard deviation over the 10 runs. The error is the distance and angular difference to the optimal position of the robot. The figure is split into two plots, the first one displays the angular error and the second one the distance error.

One can see that the angular error stayed in the range $\pm 7.5°$, except for 4 outliers towards the end. These are most likely runs in which the person took narrow turns, which are examined in more detail in scenario 2 5.2.

The distance error plot has a peak in the beginning of the simulation, this is because the simulation starts with a 1 meter distance error due to the starting position of the person. The person also starts with a velocity of 1 m/s, while the robot starts with a velocity of 0 m/s and must accelerate first. From the mean error and standard deviation it can be seen that the robot managed to do quite well with this task. The mean error is very close to 0 meters and the standard deviation is in the range of $\pm$10cm over the rest of the simulation.

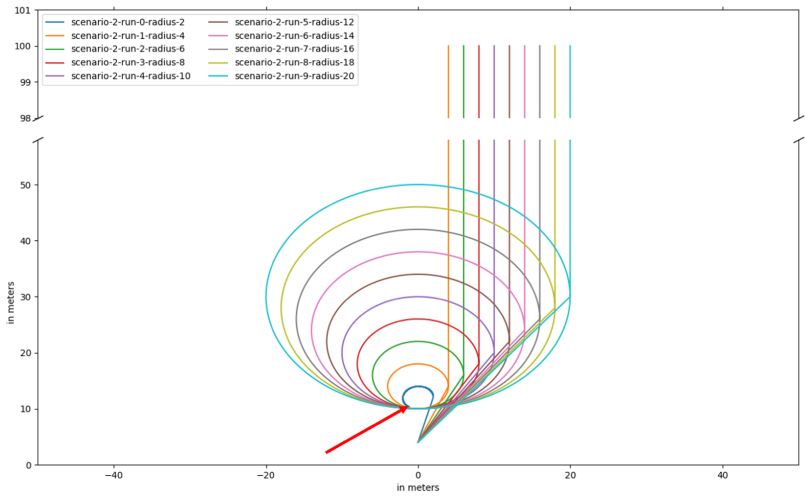


Figure 5.3: Scenario 2 - The figure shows the path walked for each run by the person. The person always starts at the same location and each run has an increasing radius. The red arrow points to the circle walked in the first run which is interrupted because the robot lost the person.

5.2 Scenario 2 - Circles

In the second experiment, the main goal was to test if the robot could keep up with the person walking in a circle. For that the person first walked in a straight line to the position from which it started to walk a circle. When the circle was completed, the person continued to walk in a straight line until it stopped 100 meters away from the original starting point in y direction 5.3.

During the whole experiment the person walked with a constant velocity of 1 m/s. In total 10 different runs were conducted and in each run the radius was given by $(run + 1) * 2$ meters, where run started with 0.

The experiment was successful, when the robot managed to keep up until the person reached the end mark.

5.2.1 Scenario 2 - Result

The robot managed to keep up with the person in every run except the first one, which had a radius of 2 meters. From the walked path of the person 5.3 one can see, that the robot lost the person after about 3/4 of the circle. Looking at 5.4, the blue line, one can see, that the angular error was too great meaning the person left the radar area. This could have resulted from the robot not being able to turn fast enough.

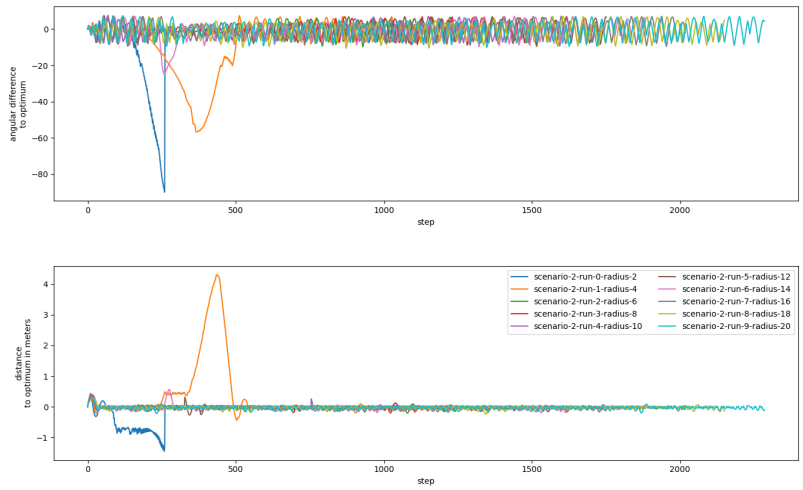


Figure 5.4: Scenario 2 - The error values created during the runs. The first run failed, because the angular error got bigger than -90°. It is noticeable, that the overall error values behaved inversely to the radius.

In the second run the angular error was also relatively big, and the distance error was even greater than in the first run. But the robot still managed to catch up with the person in the end.

In general, it is noticeable, that the angular and distance error behaved inversely to the radius. After the 4th run, when the radius was greater than 10 meters, the error values are very similar to the errors created in the first scenario during the random walks.

5.3 Scenario 3 - Person Stopping

In experiment 3 it was tested if the robot can stop if the person stops. To do so the person walked with a constant velocity of 1 m/s in front of the robot in a straight line. After a random period, within 10 to 100 steps, the person decreased its velocity within 10 steps to 0 m/s. Then it waited a random period, between 20 and 100 steps until it accelerated again within 10 steps to 1 m/s and continued to walk for another 100 steps. This experiment was also repeated 10 times.

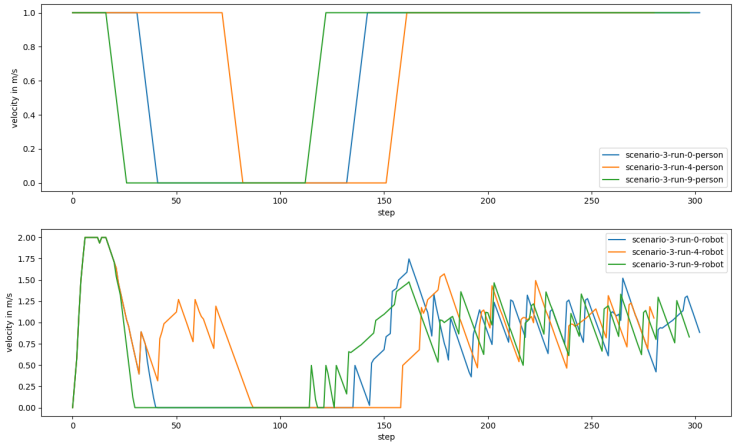


Figure 5.5: Scenario 3 - Velocity of the Person and robot during the simulation of 3 runs. The robot reacted quite fast to the person decelerating and accelerating.

5.3.1 Scenario 3 - Result

The robot managed to stop in each of the 10 runs. For better visibility figures 5.5 and 5.6 show only the velocity of the person and robot and error from three runs, but the other runs behaved similarly.

One can see from the velocity plot in 5.5 that the robot reacted quite fast and started to decelerate as soon, as the person decelerated. The same behavior can be seen when the person accelerates.

Looking at the error plot 5.6 one can see that the robot did not get further away than 1 meter from the optimal position before stopping, which was the case for two runs in total. The others runs are in the range of ± 0.25m.

The cases with the maximum distance error of 1 meter have in common, that the person stopped very early in the simulation. Because the person starts with a velocity of 1 m/s and the robot with 0 m/s, it first needs to catch up with the person. Due to that it will accelerate a lot in the beginning and decelerate as soon as it managed to catch up with the person. So, if the person starts stopping early, the velocity of the robot is still greater, because it was still in the process of catching up, and it will take the robot longer to stop. This leads to a greater distance error when the robot managed to stop.

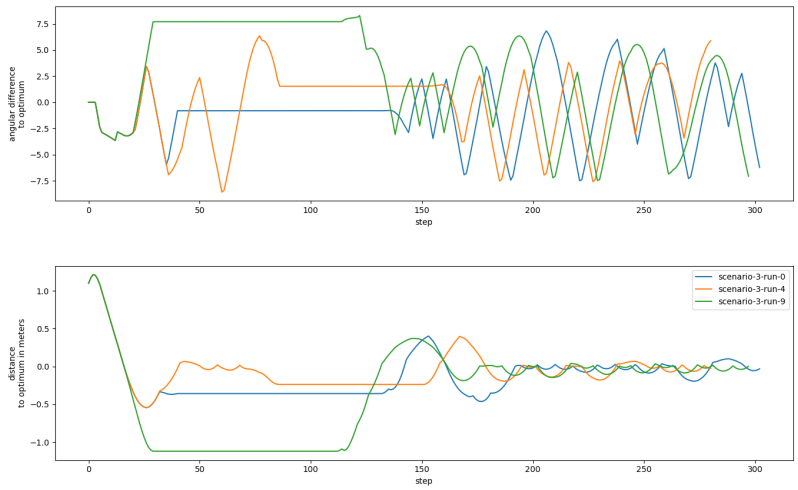


Figure 5.6: Scenario 3 - The error values created during the three runs. One can see that the distance error did not get greater than 1 meter which was the case for two runs. This is further explained i 5.3.1. The other runs are in the range of ± 0.25m.

5.4 Scenario 4 - different starting positions

The person usually stays in a small area of the CANN around the optimal position. This leads to only a small region of the network being tested. Therefore, in this experiment the person is placed at random positions within the radar area and the robot must get to the optimal position within 800 steps.

The scenario has been repeated two times, the first time the person walked a random path from the starting point and the second time the person walked in a straight line from the starting point. Both cases were repeated 60 times and the person walked with a constant velocity of 1 m/s.

The robot caught up with the person when he was less than 1 meter and $\pm 10°$ away from the optimal position.

5.4.1 Scenario 4 - Result

In figure 5.7 the starting positions are displayed where the robot is positioned at (0, 0). Plot 5.7a displays the starting positions for the random walk and plot 5.7b for the straight walk.

In the first case the robot failed 30 times, because it took too many steps, 6 times,

because the person left the visible area and succeeded 24 times.

In the second case the robot failed 6 times, because the person left the visible area for the robot and succeeded 54 times, for which it never needed more than 600 steps.

Looking at figure 5.8, which displays the mean error and standard deviation over the runs, one can see, that the distance and angular error goes against 0 in both cases. This indicates that the robot was also able in the random walk case to keep up with the person in most runs and that the walked path of the person prevented him from reaching the condition that he caught up. If the robot would have had more steps, he might have been able to catch up.

From plot b 5.8b one can see, that in the straight walking case, the robot never needed more than 526 steps to catch up with the person. This was already an exception, as most runs caught up with the person after 300-400 steps. But this is of course highly dependent on the starting position.

5.5 Scenario 5 - Robustness against Noise

In this experiment the network was tested for robustness. This was done by increasing the number of randomly placed noise spikes from the CANN every 20 steps by 2 more spikes. The network started with 2 noise spikes.

The person walked in a straight line, starting at the optimal position, with a constant velocity of 1 m/s. The experiment failed, when the person walked out of the visible are of the robot and it was repeated 60 times.

The CANN itself will emit 52 spikes in the simulation in every step, when no noise is applied.

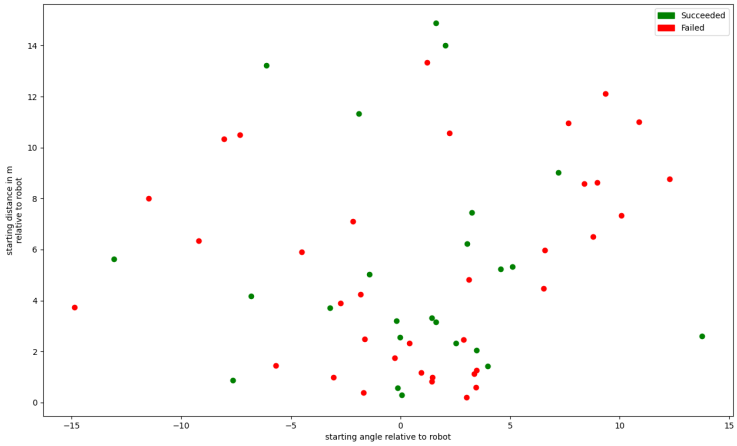
5.5.1 Scenario 5 - Result

In table 5.1 the results from the simulations are displayed. It took on average 21.48, with a median of 22, random noise spikes until the robot failed to keep up with the person. The standard deviation here was 4.009.

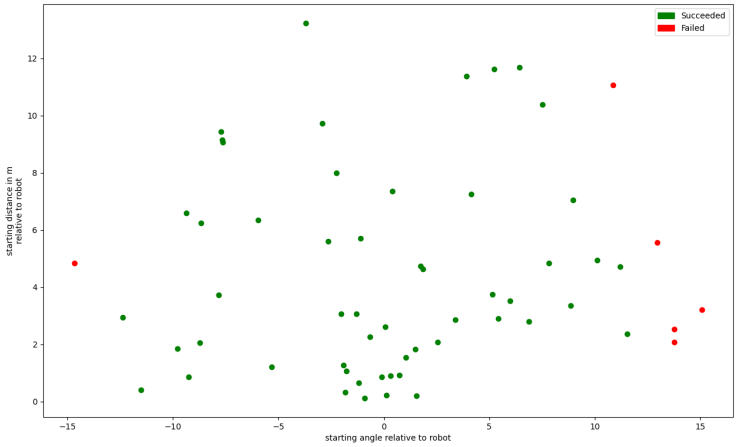
The median number of steps is 202 and the mean number of steps is 198.42. This corresponds to an average walked distance of 19.482 meters.

Considering the number of spikes without noise, the network seems quite robust against noise, as the robot can handle to follow the person until about 30% of the total number of spikes are noise.

From the violin plot 5.9 it can be seen that most of simulations failed around the mean. But the standard deviation is relatively high with 40.08, considering that it is 20% of the mean value.



(a) Starting Positions Random Walk



(b) Starting Positions Straight Walk

Figure 5.7: Scenario 4 - Figures display the starting position for each of the runs. Green dots indicate that the robot managed to catch up with the person, while red dots indicate, that he lost the person or needed too many steps.

This indicates that the position of noise spikes plays an important role for failing, as the person behaved the same in every run and the network is deterministic. This can also be seen from the outliers, where the lowest number of steps was 99 with 10 noise

Results from the simulations	
Mean number of steps until failing	198.42
Standard deviation number of steps until failing	40.08
Median number of steps until failing	202
Mean number of noise spikes before failing	21.48
Standard deviation number of noise spikes before failing	4.009
Median number of noise spikes before failing	22
Mean walked distance in meters until failing	19.842
Standard deviation walked distance in m until failing	4.008

Table 5.1: Scenario 5 - Results from the simulation

spikes, while the highest number of steps was 292 with 54 noise spikes.

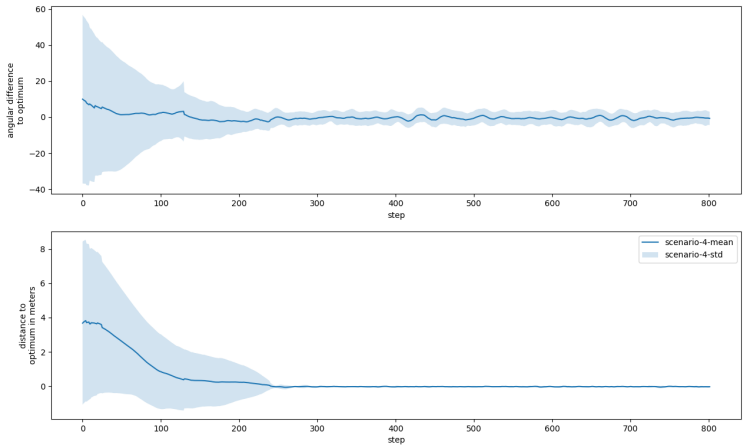
5.6 Number of Spikes

During a simulation step the number of spikes stays relatively constant. The simulated CANN will always emit 52 spikes, as the spike circle has a constant size in the simulation. This is done twice, once to the velocity neurons and once to the CANN neurons.

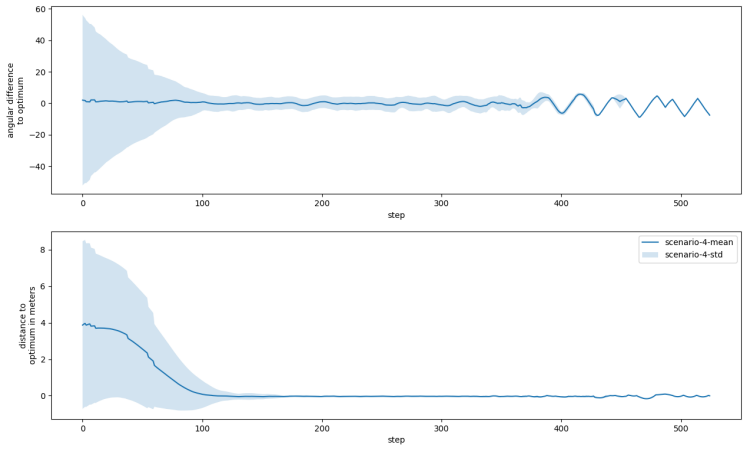
The distance network will spike 1599 times, over inhibitory synapses, to prevent other neurons from spiking. 2 inhibitory and 2 excitatory spikes are emitted to the distance neurons, which will in turn emit two output spikes to the velocity neurons if the person moved.

Only the velocity neurons can have a varying number of output spikes. Over 1000 simulation steps, they combined spiked 4.164 times on average per step.

This results in the network emitting on average 1713.164 spikes per simulation step.



(a) Mean and standard deviation of the angular and distance error for scenario 4 with random walk. One can see that both errors go against 0 over time, which indicates, that the robot managed to keep up with the person but did not get close enough to reach the defined goal.



(b) Mean and standard deviation of the angular and distance error for scenario 4 with straight walk. One can see that both errors go against 0 over time, which corresponds to the result, that the robot managed to reach its goal position most of the time. Also noticeable, the robot never needed more than 526 steps.

Figure 5.8: Scenario 4 - Mean and standard deviation

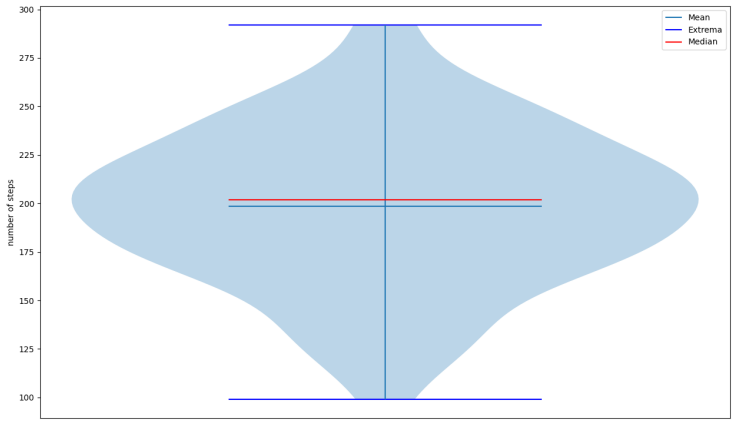


Figure 5.9: Violin Plot for scenario 5. Most runs failed around the mean, the lowest number of steps was 99 with 10 noise spikes, while the highest number of steps 292 with 54 noise spikes. This indicates that the position of spikes plays an important role.

6 Conclusion

In this thesis a SNN was developed, that controls a simulated robot to follow a person. First existing solutions for similar problems were researched, especially in the context of SNNs. Afterwards a network structure was proposed, implemented and evaluated in different scenarios.

The following sections concludes the results and proposes solutions for existing limitations.

6.1 Discussion of results

From the evaluation chapter 5 one can see, that the proposed SNN architecture to control the robot works in most cases. The scenarios were designed to test general situations, like a normal following behavior, and to find limitations with extreme situations.

The robot had problems with curvatures that had a small radius and struggled in some runs to keep up with the person, when the person was placed at random positions within the network.

In the random walk scenario 5.1 the angular error mostly stayed within $\pm 7.5°$, which was similar for other scenarios. Even though the robot always managed to keep up with the person, this is far from perfect and would lead to bad performance in a real-world robot. The distance error in contrast performed much better and stayed within a range of ± 10 cm, which is a very promising result.

It was also shown that the network can handle up to 30% of its input spikes being noise, which is a very good ratio.

In general, it can be said, that the proposed architecture works and shows promising results. Some solutions to overcome the mentioned problems are listed in the next section.

6.2 Future Work

Even though the proposed architecture works well in most tested scenarios, it is not perfect. To make the robots following behavior more stable, one could use a higher resolution of the radar data and by that a higher resolution within the CANN. This would give the network more detailed information about the persons location, through which it could react faster to position changes.

Another option could be to further limit the angular velocity of the robot in each step, this could make the driving behavior more stable, through which the angular error could most likely be reduced.

A scenario which is not considered by the network structure itself, is what happens if the person leaves the visible area of the robot or if the output neurons do not spike in a given time step.

In both cases the network does not predict a velocity change, which would lead to the robot continue driving with the same velocity.

Training the network with more iterations, with the used training schedule, could make it more robust against scenarios in which the person is placed at a random position within the network.

As of now, the network only works with a simulated CANN and has not been tested with the CANN from [Aks22]. A next step could be to combine the two network structures and evaluate the behavior on a real-world robot.

6.3 Reproducibility

The code for the network, training, simulation and evaluation can be found under `https://gitlab.lrz.de/0000000000149C02A/master-thesis`.

To get started, it is recommended to look at the README.md.

List of Figures

Bibliography

[12a] *Brian 2 documentation*. 2012. URL: https://brian2.readthedocs.io/en/stable/.

[12b] *Matplotlib: Visualization with Python*. 2012. URL: https://matplotlib.org.

[12c] *Welcome to Python.org*. 2012. URL: https://www.python.org/.

[17a] *Action potentials and synapses*. Nov. 2017. URL: https://qbi.uq.edu.au/brain-basics/brain/brain-physiology/action-potentials-and-synapses.

[17b] *Introduction to mobile robotics - SS 2017*. May 2017. URL: http://ais.informatik.uni-freiburg.de/teaching/ss17/robotics/exercises/solutions/03/sheet03sol.pdf.

[20a] *Brain neurons and synapses*. Nov. 2020. URL: https://human-memory.net/brain-neurons-synapses/.

[20b] *The Neuron*. Nov. 2020. URL: https://openbooks.lib.msu.edu/app/uploads/sites/6/2020/11/Neuron.jpg.

[Aks22] R. D. Aksoy. "RADAR based Object Tracking with Attractor Spiking Neural Networks." Master's Thesis in informatics. Technical University of Munich, Feb. 2022.

[Aug+21] D. Auge, J. Hille, E. Mueller, and A. Knoll. "A survey of encoding techniques for signal processing in spiking neural networks." In: *Neural Processing Letters* 53.6 (2021), pp. 4693–4710.

[Bin+18] Z. Bing, C. Meschede, K. Huang, G. Chen, F. Rohrbein, M. Akl, and A. Knoll. "End to end learning of spiking neural network based on r-stdp for a lane keeping vehicle." In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 4725–4732.

[Bin+19] Z. Bing, I. Baumann, Z. Jiang, K. Huang, C. Cai, and A. Knoll. "Supervised learning in SNN via reward-modulated spike-timing-dependent plasticity for a target reaching vehicle." In: *Frontiers in neurorobotics* 13 (2019), p. 18.

[BP98] G.-q. Bi and M.-m. Poo. "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type." In: *Journal of neuroscience* 18.24 (1998), pp. 10464–10472.

[CC18] I. Condes and J. M. Canas. "Person following robot behavior using deep learning." In: *Workshop of Physical Agents*. Springer. 2018, pp. 147–161.

[CD+08] N. Caporale, Y. Dan, et al. "Spike timing-dependent plasticity: a Hebbian learning rule." In: *Annual review of neuroscience* 31.1 (2008), pp. 25–46.

[Cha+20] W. P. Chan, S. Radmard, Z. Q. Hew, J. Morris, E. Croft, V. der Loos, and H. Machiel. "Autonomous Person-Specific Following Robot." In: *arXiv preprint arXiv:2010.08017* (2020).

[CST17] B. X. Chen, R. Sahdev, and J. K. Tsotsos. "Integrating stereo vision with a CNN tracker for a person-following robot." In: *International Conference on Computer Vision Systems*. Springer. 2017, pp. 300–313.

[CV95] C. Cortes and V. Vapnik. "Support-vector networks." In: *Machine learning* 20.3 (1995), pp. 273–297.

[DM21] C. K. Dewa and J. Miura. "Integrating Multiple Policies for Person-Following Robot Training Using Deep Reinforcement Learning." In: *IEEE Access* 9 (2021), pp. 75526–75541.

[Doi+12] G. Doisy, A. Jevtic, E. Lucet, and Y. Edan. "Adaptive person-following algorithm based on depth images and mapping." In: *Proc. of the IROS Workshop on Robot Motion Planning*. Vol. 20. 12. 2012.

[Dum18] G. Dumas. *Example: Izhikevich_2007*. Aug. 2018. URL: `https://brian2.readthedocs.io/en/stable/examples/frompapers.Izhikevich_2007.html`.

[DYC17] M. Do Hoang, S.-S. Yun, and J.-S. Choi. "The reliable recovery mechanism for person-following robot in case of missing target." In: *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. IEEE. 2017, pp. 800–803.

[GD43] R. Galambos and H. Davis. "The response of single auditory-nerve fibers to acoustic stimulation." In: *Journal of neurophysiology* 6.1 (1943), pp. 39–57.

[Ger+14] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[Ger+96] W. Gerstner, R. Kempter, J. L. Van Hemmen, and H. Wagner. "A neuronal learning rule for sub-millisecond temporal coding." In: *Nature* 383.6595 (1996), pp. 76–78.

[Ger17] W. Gerstner. *STDP*. Scholarpedia, Dec. 2017. URL: http://www.scholarpedia.org/article/File:STDP-Fig1.JPEG.

[GFS07] R. Gockley, J. Forlizzi, and R. Simmons. "Natural person-following behavior for social robots." In: *Proceedings of the ACM/IEEE international conference on Human-robot interaction*. 2007, pp. 17–24.

[GKR96] T. J. Gawne, T. W. Kjaer, and B. J. Richmond. "Latency: another potential code for feature binding in striate cortex." In: *Journal of neurophysiology* 76.2 (1996), pp. 1356–1360.

[GM08] T. Gollisch and M. Meister. "Rapid neural coding in the retina with relative spike latencies." In: *science* 319.5866 (2008), pp. 1108–1111.

[HM03] N. Hirai and H. Mizoguchi. "Visual tracking of human back and shoulder for person following robot." In: *Proceedings 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2003)*. Vol. 1. IEEE. 2003, pp. 527–532.

[Izh07] E. M. Izhikevich. "Solving the distal reward problem through linkage of STDP and dopamine signaling." In: *Cerebral cortex* 17.10 (2007), pp. 2443–2452.

[Jia+13] S. Jia, L. Wang, S. Wang, and C. Bai. "Fuzzy-based intelligent control strategy for a person following robot." In: *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE. 2013, pp. 2408–2413.

[Kai+16] J. Kaiser, J. C. V. Tieck, C. Hubschneider, P. Wolf, M. Weber, M. Hoff, A. Friedrich, K. Wojtasik, A. Roennau, R. Kohlhaas, et al. "Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks." In: *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*. IEEE. 2016, pp. 127–134.

[Kau+19] S. Kautsar, B. Widiawan, B. Etikasari, S. Anwar, R. D. Yunita, and M. Syai'in. "A simple algorithm for person-following robot control with differential wheeled based on depth camera." In: *2019 International Conference on Computer Science, Information Technology, and Electrical Engineering (ICOMITEE)*. IEEE. 2019, pp. 114–117.

[KM16] K. Koide and J. Miura. "Identification of a specific person using color, height, and gait features for a person following robot." In: *Robotics and Autonomous Systems* 84 (2016), pp. 76–87.

[KMM20] K. Koide, J. Miura, and E. Menegatti. "Monocular person tracking and identification with on-line deep feature selection for person following robots." In: *Robotics and Autonomous Systems* 124 (2020), p. 103348.

[LeC+89] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation applied to handwritten zip code recognition." In: *Neural computation* 1.4 (1989), pp. 541–551.

[Maa97] W. Maass. "Networks of spiking neurons: the third generation of neural network models." In: *Neural networks* 10.9 (1997), pp. 1659–1671.

[Mar+97] H. Markram, J. Luebke, M. Frotscher, and B. Sakmann. "Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs." In: *Science* 275.5297 (1997), pp. 213–215.

[Miu+10] J. Miura, J. Satake, M. Chiba, Y. Ishikawa, K. Kitajima, and H. Masuzawa. "Development of a person following robot and its experimental evaluation." In: *Intelligent Autonomous Systems 11*. IOS Press, 2010, pp. 89–98.

[Mon+08] M. A. Montemurro, M. J. Rasch, Y. Murayama, N. K. Logothetis, and S. Panzeri. "Phase-of-firing coding of natural visual stimuli in primary visual cortex." In: *Current biology* 18.5 (2008), pp. 375–380.

[MP43] W. S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[MP69] M. Minsky and S. Papert. "An introduction to computational geometry." In: *Cambridge tiass., HIT* 479 (1969), p. 480.

[OL96] N. A. Otmakhova and J. E. Lisman. "D1/D5 dopamine receptor activation increases the magnitude of early long-term potentiation at CA1 hippocampal synapses." In: *Journal of Neuroscience* 16.23 (1996), pp. 7478–7486.

[Oua+21] A. Ouaknine, A. Newson, J. Rebut, F. Tupin, and P. Pérez. "CARRADA Dataset: Camera and Automotive Radar with Range- Angle- Doppler Annotations." In: *2020 25th International Conference on Pattern Recognition (ICPR)*. 2021, pp. 5068–5075. DOI: 10.1109/ICPR48806.2021.9413181.

[Pan+20] L. Pang, Y. Zhang, S. Coleman, and H. Cao. "Efficient hybrid-supervised deep reinforcement learning for person following robot." In: *Journal of Intelligent and Robotic Systems* 97.2 (2020), pp. 299–312.

[Paz01] R. A. Paz. "The design of the PID controller." In: *Klipsch school of Electrical and Computer engineering* 8 (2001), pp. 1–23.

[Por+16] G. Portelli, J. M. Barrett, G. Hilgen, T. Masquelier, A. Maccione, S. Di Marco, L. Berdondini, P. Kornprobst, and E. Sernagor. "Rank order coding: a retinal information decoding strategy revealed by large-scale multielectrode array retinal recordings." In: *eneuro* 3.3 (2016).

[PSD19] F. Paredes-Vallés, K. Y. Scheper, and G. C. De Croon. "Unsupervised learning of a hierarchical spiking neural network for optical flow estimation: From events to global motion perception." In: *IEEE transactions on pattern analysis and machine intelligence* 42.8 (2019), pp. 2051–2064.

[Ren+16] Q. Ren, Q. Zhao, H. Qi, and L. Li. "Real-time target tracking system for person-following robot." In: *2016 35th Chinese Control Conference (CCC)*. IEEE. 2016, pp. 6160–6165.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors." In: *nature* 323.6088 (1986), pp. 533–536.

[RJP19] K. Roy, A. Jaiswal, and P. Panda. "Towards spike-based machine intelligence with neuromorphic computing." In: *Nature* 575.7784 (2019), pp. 607–617.

[Ros58] F. Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[SCM12] J. Satake, M. Chiba, and J. Miura. "A SIFT-based person identification using a distance-dependent appearance model for a person following robot." In: *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE. 2012, pp. 962–967.

[SKC99] H. Sidenbladh, D. Kragic, and H. I. Christensen. "A person following behaviour for a mobile robot." In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*. Vol. 1. IEEE. 1999, pp. 670–675.

[SM09] J. Satake and J. Miura. "Robust stereo-based person detection and tracking for a person following robot." In: *ICRA Workshop on People Detection and Tracking*. 2009, pp. 1–10.

[SMA00] S. Song, K. D. Miller, and L. F. Abbott. "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity." In: *Nature neuroscience* 3.9 (2000), pp. 919–926.

[TFM96] S. Thorpe, D. Fize, and C. Marlot. "Speed of processing in the human visual system." In: *nature* 381.6582 (1996), pp. 520–522.

[Wan+09] X. Wang, Z.-G. Hou, M. Tan, Y. Wang, and L. Hu. "The wall-following controller for the mobile robot using spiking neurons." In: *2009 international conference on artificial intelligence and computational intelligence*. Vol. 1. IEEE. 2009, pp. 194–199.

[WH60] B. Widrow and M. E. Hoff. *Adaptive switching circuits*. Tech. rep. Stanford Univ Ca Stanford Electronics Labs, 1960.

[Wnu] A. Wnuk. *How inhibitory neurons shape the brain's code*. URL: https://www.brainfacts.org/brain-anatomy-and-function/cells-and-circuits/2021/how-inhibitory-neurons-shape-the-brains-code-100621.

[Wu+16] S. Wu, K. M. Wong, C. A. Fung, Y. Mi, and W. Zhang. "Continuous attractor neural networks: candidate of a canonical model for neural information representation." In: *F1000Research* 5 (2016).