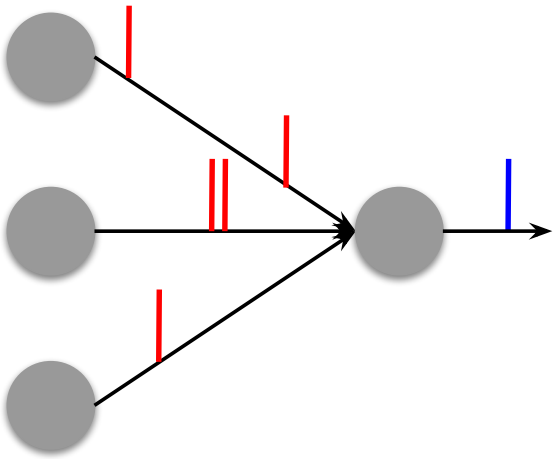

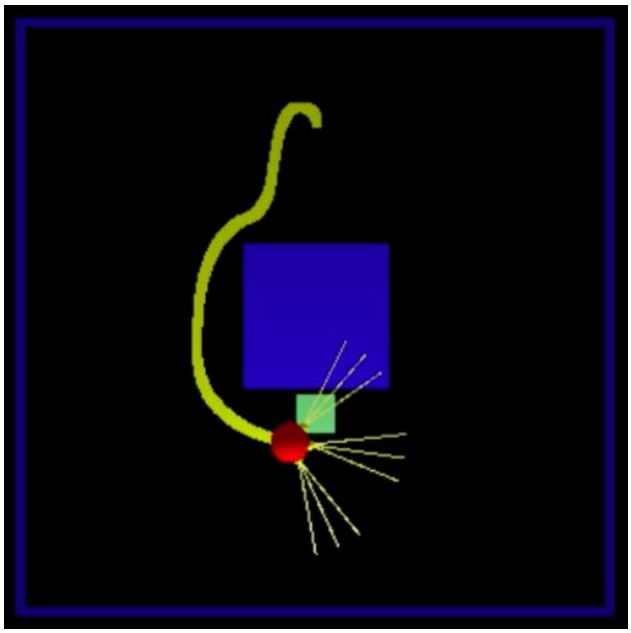

Spiking neural network for targeted navigation and collision avoidance in an autonomous robot

Master's thesis in Complex Adaptive Systems

MALIN RAMNE

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
DIVISION OF VEHICLE ENGINEERING AND AUTONOMOUS SYSTEMS (VEAS)

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020
www.chalmers.se

MASTER'S THESIS IN COMPLEX ADAPTIVE SYSTEMS

Spiking neural network for targeted navigation and collision avoidance in an autonomous robot

MALIN RAMNE

Department of Mechanics and Maritime Sciences
Division of Vehicle engineering and autonomous systems (VEAS)
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2020

Spiking neural network for targeted navigation and collision avoidance in an autonomous robot
MALIN RAMNE

Master's thesis 2020:36
Department of Mechanics and Maritime Sciences
Division of Vehicle engineering and autonomous systems (VEAS)
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
Spiking neuron model and example of robot trajectory

Chalmers Reproservice
Göteborg, Sweden 2020

Spiking neural network for targeted navigation and collision avoidance in an autonomous robot
Master's thesis in Complex Adaptive Systems
MALIN RAMNE
Department of Mechanics and Maritime Sciences
Division of Vehicle engineering and autonomous systems (VEAS)
Chalmers University of Technology

Abstract

In this project a two-layered spiking neural network is implemented for targeted navigation and collision avoidance in an autonomous robot. The performance can be further improved by training with RM-STDP. The proposed modified STDP for inhibitory synapses yields even better performance in the networks and often after fewer rounds of training. The project also presents an evolutionary algorithm for determining the synaptic connections of a spiking neural network with a hidden layer. The evolutionary algorithm shows potential of working as a tool for determining the synaptic connectivity of spiking neural networks, however this project only explores a first rather simple implementation of the algorithm. In one run of the algorithm, with networks with a hidden layer consisting of 100 hidden neurons, a network with the ability to arrive at most of the test cases was evolved in less then 20 generations. However further work is necessary in order to determine the true potential of this approach.

The performance of the spiking neural networks in this project are compared with the performance of non-spiking neural networks in order to determine advantages and disadvantages of using spiking neural networks in this specific case. The non-spiking neural networks perform better on the target navigation and collision avoidance tasks than corresponding spiking neural networks. The spiking neural network with no hidden layer arrived at the target in 842 out of 1000 test cases, and the non-spiking network arrived in 914 cases. With the hidden layer the number or arrivals where 784 for the spiking network and 916 for the non-spiking. This indicates that the advantages of the spike-rate interpretation of non-spiking neurons outweigh the expected advantages of encoding information in the specific timing of spikes in spiking neural networks, at least in the network structures and the tasks examined in this project.

Keywords: Spiking neural network, spike timing dependent plasticity, reward modulation, evolutionary algorithm

Preface

The work with this thesis started early in 2020 and it is now being finalized in early June. This period of time will likely go down in history, however not because of any great revelations in this thesis, but rather due to the virus-pandemic that has spread across the globe. Thanks to the technical advancements of the past few decades I have been able to continue working from home throughout this spring. I would like to express my gratitude to my supervisor Mattias Wahde. Thank you for your help, guidance and valuable insight in this project, and also for the very interesting and inspiring courses that you have held and that I have had the privilege of participating in during my education. I would also like to thank my family, both humans, horses, dogs and cats, for all your support during this project and throughout my whole education.

Malin Ramne, Gothenburg, May 2020

Contents

1 Introduction

Spiking neural networks are a type of artificial neural networks that are inspired by biological neural networks. As opposed to traditional artificial neural networks, where all neurons fire synchronously, the neurons of spiking neural networks fire asynchronously when their individual inputs have reached a certain threshold. The signals passed between the neurons are short pulses allowing for incorporation of spatial-temporal information that is lost in other types of artificial neural networks. The similarity to biological neurons indicates that spiking neural networks might be suitable for a wide array of applications, especially where fast execution is required and where energy resources are scarce, such as in robotics.

There are several examples where spiking neural networks have been implemented for robot control, [3] provides a review of what has been done in the field up until 2018. Most implementations have network architectures, neuron models, reward models and learning rules that work well for the specific task at hand, and the implementations vary a lot between different projects. There are still no general purpose and easy-to-use spiking neural network-based controls, as there are for many applications of non-spiking neural networks, due to the complexity in learning, tuning parameters and defining rewards. In several projects concerning spiking neural networks for navigation and collision avoidance careful thought has been put into the structure and connectivity of the networks to achieve the desired behaviour, see [4], [15], [18] and [19]. In this project an attempt is made to achieve similar results but with less thought put into the actual structure of the networks. Instead the synaptic connections are determined using an evolutionary algorithm. In the evolutionary algorithm the fitness of each individual is determined by the networks ability to navigate the robot to a target without collisions. Evolutionary algorithms have been used to perform various tasks related to artificial neural networks, such as training, network architecture design and learning rule adaptation [20]. When it comes to spiking neural networks for robot control there are examples where evolutionary algorithms have shown promising results regarding determining network structures and training networks [1], [2], [5].

In this project the evolutionary algorithm is combined with the biologically plausible reward modulated spike timing dependent plasticity, RM-STDP, for training the networks. A slightly simpler approach has been taken when it comes to the STDP than in some previous projects, such as [4] and [15]. In this project the reward that is administered to the network is equal for all synapses in the network, whereas in other projects the reward has been different for different synapses. The reasoning behind this simplified approach is that the reward in biological neural networks is in the form of dopamine being released in the vicinity of the neurons. Even if the dopamine may not released equally to the whole network (it is a highly complicated process) the dose is also likely not tweaked to an exact value for each individual synapse. The resulting change in synaptic strength is thus in this project set to be a function of the global reward (corresponding to a global concentration of dopamine) and the synaptic eligibility trace, a form of short term measure of the pre- and postsynaptic spikes.

1.1 Purpose

The purpose of this project is to implement a spiking neural network in a robot and train it to execute tasks such as targeted navigation. The aim is to explore whether spiking neural networks can be used to give an autonomous robot the ability to navigate to a specific goal while avoiding collisions with fixed obstacles in its environment. The project also aims to examine whether an evolutionary algorithm can be used to determine the structure of such networks. The performance of the spiking neural networks are compared with the performance of more traditional artificial neural networks in order to determine advantages and disadvantages of using spiking neural networks in this specific case.

1.2 Limitations

The spiking neural networks have only been implemented in a simulated robot. Translation into a real-world robot may very well be possible, but that is outside the scope of this project. Furthermore the environment that the robot is set to navigate through is two-dimensional and relatively simple. The robot receives input from its surroundings from simulated IR-sensors and has information about its own location and heading from a simulated odometer and compass.

2 Theory and methods

This section presents the theory behind the neural network models and the simulated robot as well as descriptions of the methods implemented for training and setting the network weights.

2.1 Artificial neural networks

Artificial neural networks are a form of computational system inspired by biological neural networks. They consist of computational units, so called *artificial neurons*, that are connected to each other to form networks. By altering the connections between the artificial neurons the networks can learn to perform certain tasks. The simplest neuron model used in artificial neural networks is the *McCulloch-Pitts* neuron, sometimes also referred to as threshold gate, which maps real-valued inputs to binary output. This is done by summing the weighted inputs and comparing them to a threshold, as illustrated in Figure 2.1. This neuron model can be modified to map real-valued inputs to a real-valued output by applying an activation function to the weighted sum of inputs instead of comparing the sum to a threshold. A *sigmoid function* is commonly used as activation function.

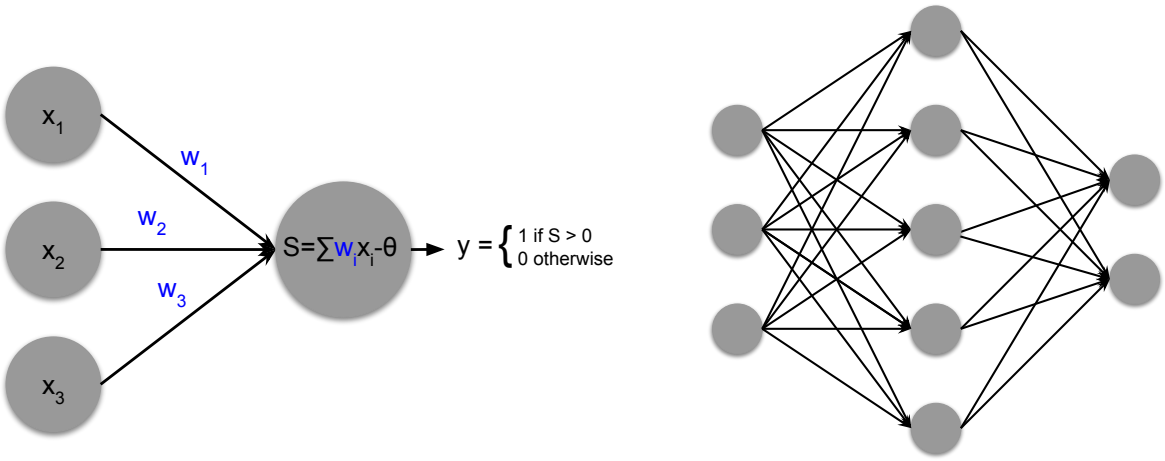

Figure 2.1: *Left: McCulloch Pitts neuron model, the weighted inputs are summed and compared to the threshold θ. If the weighted sum is larger than the threshold the neuron outputs 1, otherwise 0. Right: typical feedforward neural network structure in a network with three input neurons, one hidden layer with five neurons and two output neurons. Information propagates unidirectionally from the input neurons to the output neurons.*

In Figure 2.1 the commonly used *feedforward* network structure is illustrated, where the neurons are organized in layers and information travels unidirectionally from the input neurons (left), via the hidden neuron layers, to the output neurons (right). The connections between neurons are called *synapses* and can be represented mathematically through a connectivity function w_{ij} between neurons i and j. If $w_{ij} > 0$ the synapse is said to be *excitatory*, since it contributes to exciting the neuron it connects to. If $w_{ij} < 0$ the synapse is *inhibitory*. In the simplest synapse model w_{ij} is a constant and is often referred to as the *synaptic weight*. However, experiments have shown that the connectivity w_{ij} can change over time [7]. This is called synaptic plasticity and in network theory the process of adapting the synaptic weights (and other network parameters) is called *learning*, since the adjustments may impact the network's performance for a given task. The procedure that governs the adjustment the weights is called the *learning rule*. Most synaptic models are inspired by Hebb's postulate that is often summarized as "neurons that fire together, wire together". For a network with real-valued neurons this can be formulated mathematically as

$$\Delta w_{ij} = \eta x_i y_j \tag{2.1}$$

where $x_i = w_{ij} y_i$ is the input from neuron i to neuron j scaled with the weight w_{ij}, y_j is the output of neuron j and η is the learning rate.

2.2 Spiking neural networks

A biological interpretation of a network with real-valued neurons is that the output of the activation function corresponds to the current firing rate of a biological neuron. However, it has been shown that information may be lost when only the firing rate, and not the specific firing times, of neurons are considered [12]. It has also been shown that certain tasks and computations can be carried out in biological neural networks in 20-30 ms even when the firing frequencies of the neurons involved are below 100 Hz. If only the firing rate had been considered at least 20-30 ms would be needed just to sample the firing rate, and then additional time would be needed to perform the actual computations. Thus the time required to perform the corresponding computation in this type of neural network would have to be significantly longer than in a corresponding biological neural network.

In order to achieve behaviour more similar to that of biological neural networks an artificial neural network model using spiking neurons as the computational units has been introduced, since the signals passed between biological neurons consist of short electrical pulses. If the total input to a neuron reaches a certain threshold a so called *action potential* or *spike* is generated and passed to the connecting neurons. There exist several different mathematical models of spiking neurons, often referred to as integrate-and-fire neurons. Even though most of the models used in spiking neural networks are simplifications of the truly complex behaviour or biological neurons, these neuron models are still significantly more realistic than the binary or real-valued neuron models. In particular they better describe the output of biological neurons, thus allowing for the timing of single action potentials to encode information.

2.2.1 Leaky integrate-and-fire neuron model

The leaky integrate-and-fire neuron model, or LIF model for short, is based on a summation process (integration) of the postsynaptic potentials, a mechanism that fires action potentials when the membrane potential reaches a threshold level and finally a leak process representing the cell membrane potential's decay over time in the absence of input [7]. The model makes use of the fact that it is the timing and number of neuronal action potentials that encodes information, and not the exact shape of them. Thus the action potentials can be reduced to events that happen at precise moments in time and no attempt is made to describe the exact shape of the action potentials of biological neurons. In this model the evolution of the membrane potential, $u(t)$ is described by a linear differential equation and the relation between any injected current $I(t)$ in the cell and the membrane potential is based on elementary laws from the theory of electricity. The cell membrane is represented by a capacitor C and a resistor R connected in parallel. When there is no input the membrane potential is at the resting value u_{rest}. Thus, for the input current $I(t)$, the membrane potential can be modelled with the following linear differential equation:

$$\tau_m \frac{du}{dt} = -[u(t) - u_{\text{rest}}] + RI(t) \tag{2.2}$$

where $\tau_m = RC$ is called the *membrane time constant* of the neuron. Solving the differential equation shows that the membrane potential decays exponentially to its resting value in absence of input and that τ_m is the characteristic time of decay. This concludes the leak and the integrate part of the model, all that is left to define is the firing. The *firing time* of a neuron is defined as a time t^* such that $u(t^*) = \nu$, where ν is the threshold of the the neuron. At such times a spike is emitted from the neuron, which is represented by the Dirac δ function, and immediately after t^* the potential is reset to a value $u_r < \nu$. During a refractory period of time τ_r after the spike is emitted the neuron is unaffected by action potentials of connecting neurons (all input currents are set to 0 in equation 2.2). For all times $t \neq t^*$ the membrane potential is described by equation 2.2. The characteristic behaviour of the LIF-neuron is illustrated in Figure 2.2.

2.2.2 Spike-timing-dependent plasticity

Experiments have shown that the change Δw of the synaptic weight w can be described as a function of the difference in the firing times of the the pre- and postynaptic neurons, $\Delta t = t^*_{post} - t^*_{pre}$ [7]. If the difference in time is positive the synapse is strengthened, in accordance with Hebb's postulate: if a presynaptic neuron repeatedly contributes to the firing of a postsynaptic neuron the connection between them is strengthened, where a presynaptic neuron is deemed to have contributed to the firing of a postsynaptic neuron if it has

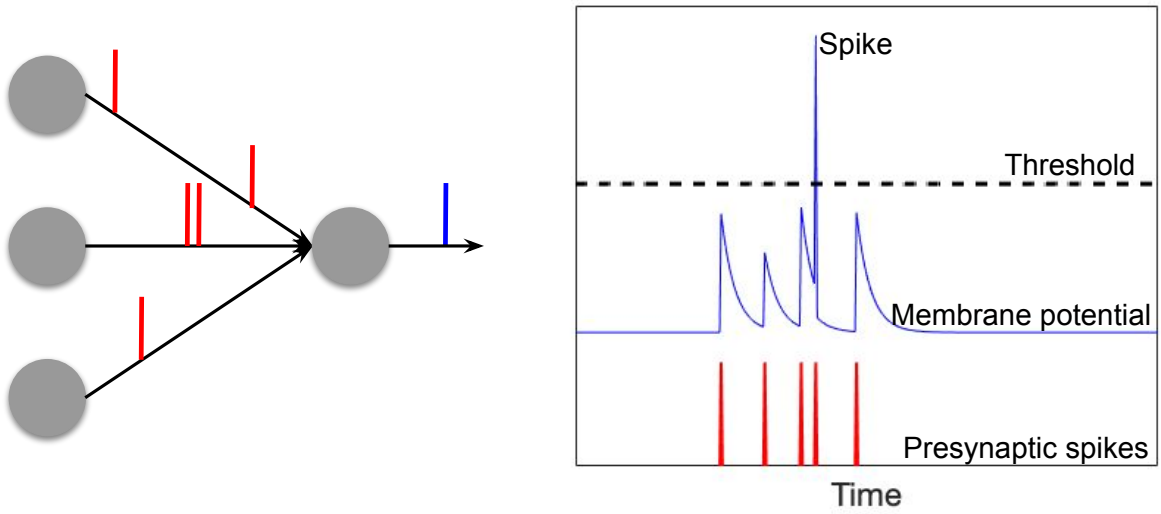

Figure 2.2: *The cell membrane potential increases (or decreases if the synapse is inhibitory) on the arrival of each presynaptic spike. When the membrane potential reaches a threshold value a spike is emitted. In abscence of input the membrane potential is described by equation 2.2.*

been active shortly before the firing of the postsynaptic neuron. If the difference in time on the other hand is negative the synapse is weakened. This is called spike-timing-dependent plasticity, abbreviated STDP. A simple model for the update of the synaptic weight is

$$\begin{aligned} \Delta w_+ &= A^+ e^{-|\Delta t|/\tau_{pre}} \text{ at } t^*_{post} \text{ if } t^*_{pre} < t^*_{post}, \\ \Delta w_- &= A^- e^{-|\Delta t|/\tau_{post}} \text{ at } t^*_{pre} \text{ if } t^*_{pre} > t^*_{post} \end{aligned} \tag{2.3}$$

where A^+ and A^- are constants and τ_{pre} and τ_{post} determine the temporal window over which the STDP is active. Figure 2.3 illustrates how the weight update depends on $\Delta t = t^*_{post} - t^*_{pre}$.

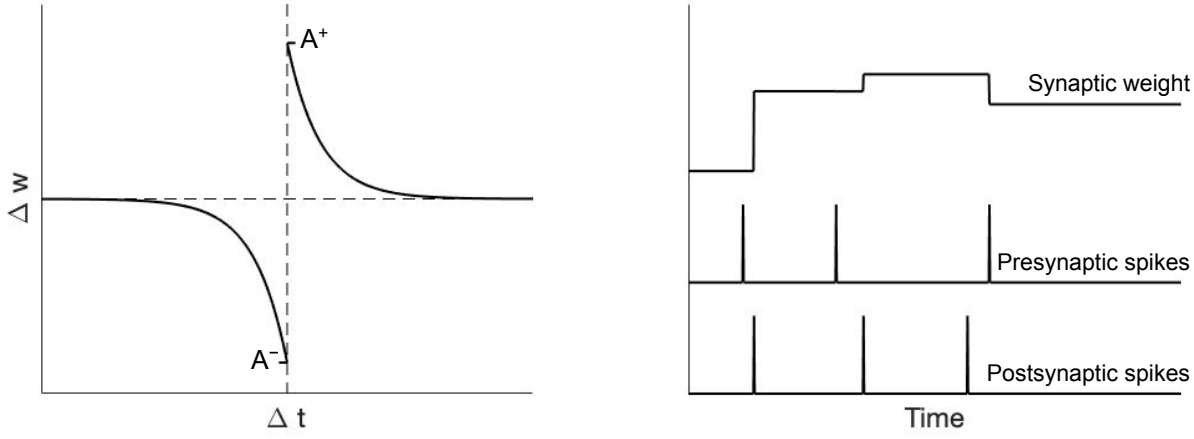

Figure 2.3: *The synaptic weight update Δw depends on the time difference Δt between the pre- and postsynaptic spikes according to equation 2.3.*

In a *pair-based* model of STDP the weight updates are calculated and applied for a specified set of pairs of pre- and postynaptic spikes. For example one could compute the update for all pairs or one could consider only the nearest presynaptic spike for each postsynaptic spike.

STDP is a form of unsupervised learning, which in general is viewed to not be sufficient when it comes to learning specific tasks and behaviours. In the biological brain the success of a certain firing pattern in the network leads to the release of chemicals such as dopamine that increase the strength of synapses. However this release of dopamine often comes with a significant delay, thus some kind of short-term memory of the neurons' actions is required. This can be achieved by keeping track of the *synaptic eligibility trace* e_{ij} in each synapse [10]. The eligibility trace e_{ij} is a form of temporary measure of the events where neuron i triggers neuron j. If the activity between the neurons stops the eligibility trace decays to zero with a time constant τ_e. In a biological sense the eligibility trace corresponds to some enzyme that regulates the plasticity of the synapse. In simulations the eligibility trace can be computed using two additional traces: the *pre-* and *postsynaptic traces* a_{pre} and a_{post}. The three traces are defined as

$$\frac{de_{ij}}{dt} = -\frac{e_{ij}}{\tau_e}, \tag{2.4}$$

$$\frac{da_{pre}}{dt} = -\frac{a_{pre}}{\tau_{pre}}, \tag{2.5}$$

$$\frac{da_{post}}{dt} = -\frac{a_{post}}{\tau_{post}}, \tag{2.6}$$

$$\text{On presynaptic spike: } a_{pre} \to a_{pre} + A^+ \tag{2.7}$$

$$e_{ij} \to e_{ij} + a_{post} \tag{2.8}$$

$$\text{On postsynaptic spike: } a_{post} \to a_{post} + A^- \tag{2.9}$$

$$e_{ij} \to e_{ij} + a_{pre} \tag{2.10}$$

Finally the synaptic weight is updated as

$$\frac{dw_{ij}}{dt} = e_{ij} \tag{2.11}$$

The three different traces resulting from pre- and postsynaptic spike trains and the resulting change in the synaptic weight w are visualized in Figure 2.4.

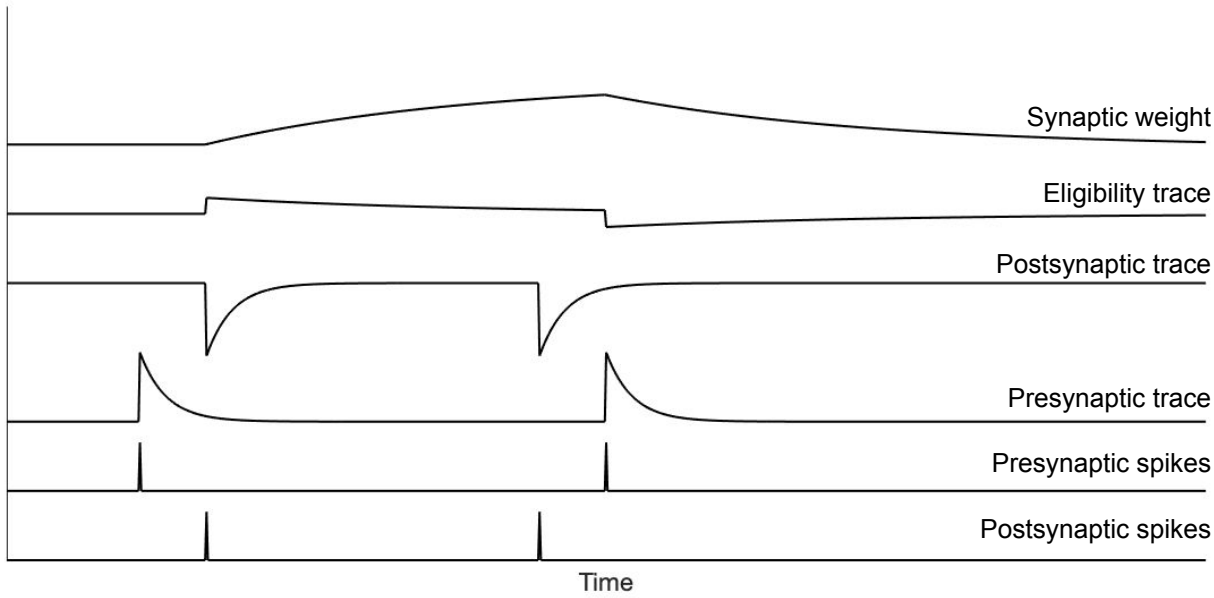

Figure 2.4: *Visualiation of STDP. When a postsynaptic spike follows shortly after a presynaptic spike the synaptic weight increases. When instead the postsynaptic spike occurs before the presynaptic spike the synaptic weight decreases, in accordance with Hebb's rule.*

The release of dopamine can be simulated by adding reward modulation to the STDP model. In the reward-modulated STDP model, RM-STDP, the update of the synaptic weights is the synaptic eligibility trace scaled

with a modulatory success signal M, corresponding to the extracellular concentration of dopamine. Although this synaptic plasticity model is based on theoretical considerations there are experimental results that support the role of eligibility traces or similar mechanisms in synaptic plasticity [6].

2.2.3 Proposed STDP for inhibitory synapses

In the pair-based STDP model described in Section 2.2.2 the synaptic weight update Δw depends on the time difference Δt between pre- and postsynaptic spikes where $\Delta t \to 0_+$ results in a larger positive change and $\Delta t \to \pm\infty$ makes $\Delta w \to \pm 0$. This model agrees well with Hebb's rule "neurons that fire together, wire together". However for inhibitory synapses (i.e where $w < 0$) in a network without any inhibitory neurons this rule does not have the desired effect. The function of an inhibitory synapse is to *inhibit* neurons from firing, thus the rule for inhibitory synapse should be something along the lines of "neurons that *don't* fire together, wire together". One way that this can be achieved is by using the following update rule:

$$\frac{dw}{dt} = A^-_{inh}(1 - e^{-\Delta t/\tau_{inh}}) + A^+_{inh}e^{-\Delta t/\tau_{inh}} \tag{2.12}$$

where $A^-_{inh} < 0$, $A^+_{inh} > 0$ are constants and τ_{inh} determines the temporal window over which the STDP is active. With this update rule the synapse is weakened ($\Delta w > 0$) if a postsynaptic spike occurs shortly after a presynaptic spike, but strengthened ($\Delta w < 0$) if the the time difference is larger or if there is no postsynaptic spike at all. In Figure 2.5 the synaptic update Δw is plotted as a function of the time difference Δt.

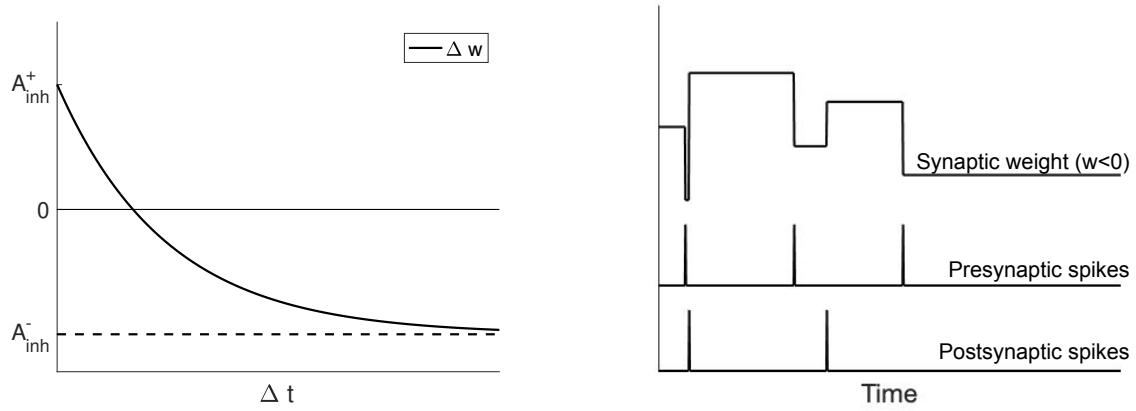

Figure 2.5: *The proposed STDP update rule for inhibitory synapses. The synapse is weakened ($\Delta w > 0$) if a postsynaptic spike occurs shortly after a presynaptic spike, but strengthened ($\Delta w < 0$) if the the time difference is larger or if there is no postsynaptic spike at all.*

Just as in the pair-based STDP model the synaptic weight update caused by dopamine release can be modeled using a *synaptic eligibility trace*, e_{ij}. The eligibility trace can in turn be computed using a *presynaptic trace*, a_{pre}. This yields the following update rule

$$\frac{de_{ij}}{dt} = -\frac{e_{ij}}{\tau_e}, \tag{2.13}$$

$$\frac{da_{pre}}{dt} = -\frac{a_{pre}}{\tau_{pre}}, \tag{2.14}$$

$$\text{On presynaptic spike: } a_{pre} \to a_{pre} + A^-_{inh}, \tag{2.15}$$

$$e_{ij} \to e_{ij} + A^-_{inh} \tag{2.16}$$

$$\text{On postsynaptic spike: } e_{ij} \to e_{ij} + \Big(\frac{A^+_{inh}}{A^-_{inh}} - 1\Big)a_{pre} \tag{2.17}$$

$$\frac{dw_{ij}}{dt} = e_{ij} \tag{2.18}$$

The traces resulting from pre- and postsynaptic spike trains and the resulting change in the synaptic weight w are visualized in Figure 2.6.

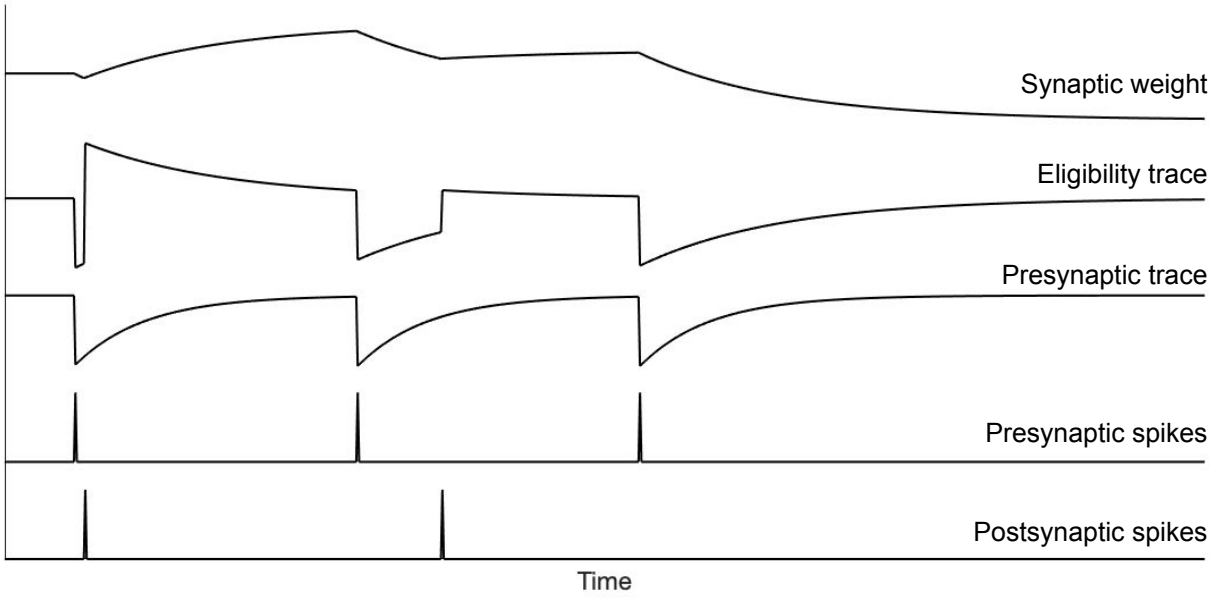

Figure 2.6: *Visualiation of the proposed STDP for inhibitory synapses. When a postsynaptic spike follows shortly after a presynaptic spike the synaptic weight is weakened ($\Delta w > 0$). When the time difference is longer or when no postsynaptic spike follows the presynaptic spike the weight is strengthened ($\Delta w < 0$).*

2.3 Robot simulation and dynamics

The simulated robot in this project has a rather simple design with a circular body with radius R and two independently driven wheels. The design of the robot along with the simulations and dynamics are inspired by [16]. To make a transition into a real life robot as simple as possible the simulations of the different parts and characteristics of the robot have been made with a corresponding real life robot in mind, although with certain simplifications.

2.3.1 Kinematics and dynamics

It is assumed that the wheels of the robot roll without slipping. Given that the robot is a rigid body and that the wheels can only move in the direction perpendicular to the wheel axis the kinematics of the entire robot, i.e the variation of the direction of motion φ and of the position of the robot (given by coordinates x and y), can be determined by the speeds v_L and v_R of the left and right wheel. Letting $\omega = \dot{\varphi}$ denote the rotation of the robot around the instantaneous center of rotation and letting v denote the speed of the center-of-mass of the robot the following relations hold:

$$v = \frac{v_L + v_R}{2}, \tag{2.19}$$

$$\omega = -\frac{v_L - v_R}{2R}. \tag{2.20}$$

These kinematics equations determine the range of possible motions for a two-wheeled differentially steered robot, however they say nothing about how the actual motion is achieved. For that we need to consider the dynamics of the robot; how the robot responds to forces and torques acting on it. The motors generate torques, τ_L and τ_R, that propel the wheels forward. The friction between the wheel and the ground then put the robot

into motion. The relation between the motor torques and the motion of the robot are given by Newton's laws of motion. However, due to losses caused by friction, limited strength of the motors and other factors, the speed and rotational velocity of the robot will be limited. The differential equations describing the dynamics of the robot should therefore also include damping terms. An approximation using linear damping, and a few simplifications of the relations, yield the following equations of motion for the robot:

$$M\dot{V} + \alpha v = \frac{\tau_L + \tau_R}{r}, \tag{2.21}$$

$$\overline{I}\ddot{\varphi} + \beta\dot{\varphi} = \frac{R(\tau_R - \tau_L)}{r}, \tag{2.22}$$

where M is the mass of the robot, r is the radius of the wheels, $\overline{I}$ denotes robot's the moment of inertia and α and β are constants related to the damping. The torque τ from each motor is given by

$$\tau = G\big(\tau_g - f_C \mathrm{sgn}(\omega) - f_v\omega\big), \tag{2.23}$$

$$\tau_g = \frac{c_t(V - c_e\omega)}{r_A}, \tag{2.24}$$

where G is the gear ratio, f_C the Coulomb friction constant, f_v the viscous friction constant, c_t a torque constant, c_e an electrical constant, r_A is the armature resistance and V is the voltage applied to the motor. The motors take as input a signal $s \in [0, 1]$ which is then scaled to an applied voltage as $V = sV_{\max}$. These are the equations at the foundation of the simulations of the robot.

2.3.2 Sensors

The robot is equipped with an odometer, a compass and IR-sensors. The signals from all of these sensors need to be simulated in a realistic way in order to make a translation into a real robot as simple as possible.

Odometer

In order for the robot to navigate around an arena it is of course crucial that the robot somehow has information about it's own location in said arena. One way of estimating this is through *odometry*, in which a wheel encoder is used to approximate the location of the robot based on the rotation (and thereby velocity) of the wheels. At the start of each simulation the odometer is calibrated to the actual position of the robot. As the robot moves the estimated position is updated based on the speed and angular speed of the robot with $\mathcal{N}(0, \sigma_o^2)$-noise, where $\sigma_o = 0.0002$, along with the estimated position in the previous step.

Compass

The heading of the robot is estimated using a compass. The compass reading is simulated as the actual heading of the robot with $\mathcal{N}(0, \sigma_c^2)$-noise, where $\sigma_c = 0.05$.

IR-sensors

Infrared proximity sensors, abbreviated IR-sensors, are ray-based sensors where a number of rays are sent out from the sensor in order to detect nearby objects. The direction β_i of each ray is determined by the angle α of the sensor relative to the heading of the robot (φ), the opening angle γ of the sensor and the number N of rays in the sensor. The reading ρ_i of each ray is modeled as

$$\rho_i = \min\left(\left(\frac{c_1}{d_i^2} + c_2\right)\cos\kappa_i, 1\right) \tag{2.25}$$

where c_1 and c_2 are constant coefficients, d_i is the distance to the nearest obstacle along the direction of ray i and κ_i is angle of ray i relative to the sensor angle α. If d_i is larger then the range of the sensor ρ_i is set to zero. The reading of the entire sensor is then given by

$$S = \frac{1}{N}\sum_{i=1}^{N}\rho_i \tag{2.26}$$

This model of the IR-sensor does not take into account the different reflectivity of different materials or the orientation of the obstacles surface.

2.4 Evolutionary algorithm for optimization

Evolutionary algorithms are a form of stochastic optimization method based on Charles Darwin's evolutionary theory. The fundamental concepts of the algorithms are that there exists a population of individuals of the same species that undergoes a gradual, hereditary change over several generations due to environmental pressure [17]. The changes are realized through mechanisms such as selection, reproduction, recombination and mutation. In order for properties of individuals to be transmitted from one generation to the next the properties of each individual have to be able to be encoded. The encoding of the properties are referred to as *chromosomes*. For selection to be possible each individual in the population has to be assigned a *fitness*, a measure of the individual's degree of adaptation to the environment that it is in. The exact details and mechanisms used in an evolutionary algorithm vary depending on the optimization problem. Figure 2.7 shows a schematic view over the evolutionary algorithm used for optimizing the neural networks in this project. The different steps of the evolutionary algorithm are described in further detail in the following sections.

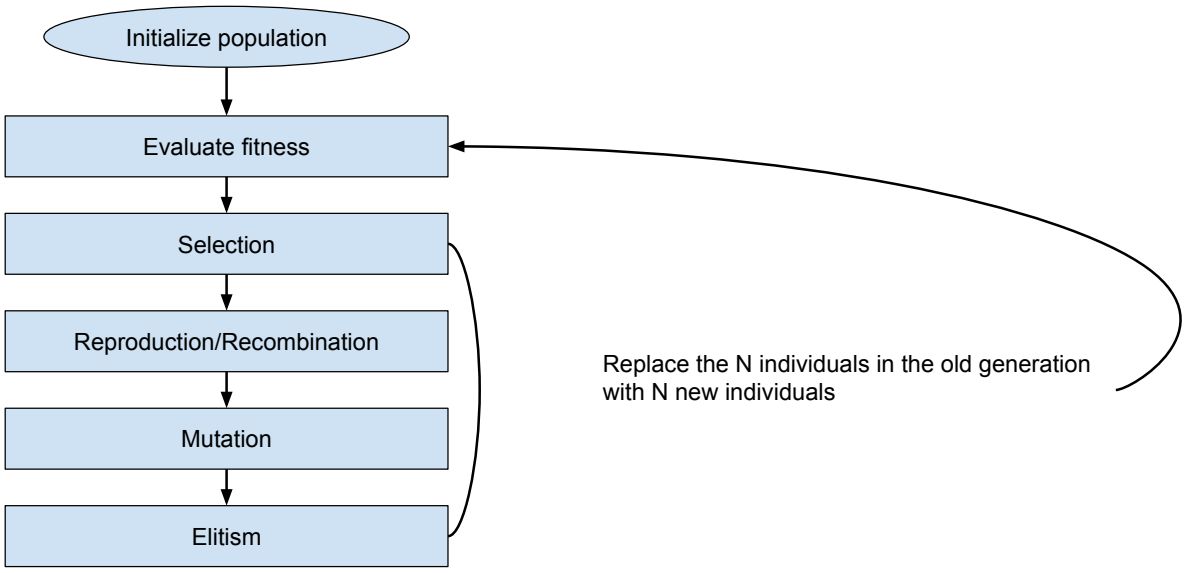

Figure 2.7: *A schematic overview of the evolutionary algorithm used to optimize the neural networks.*

Representing networks as chromosomes

In this project the evolutionary algorithm is used to optimize the layout of inhibitory and excitatory synaptic weights of the three-layered neural network controlling the robot. More specifically it is the weights from the input layer to the hidden layer and the weights from the hidden layer to the output layer that are being considered. The figure below shows how the network weights are encoded into a chromosome and how a chromosome is decoded into a network. Each weight represents a single gene in the chromosome.

Fitness

The fitness of an individual is a measure of how well adapted the individual is to the environment it is in. The point of the evolutionary algorithm for optimization is to find the individual with the highest possible fitness. It is therefore important to choose a fitness measure that accurately represents the properties and behaviours that are being optimized for. In this specific case the networks are being optimized for making the robot reach the target in the shortest time possible, therefore the fitness measure is chosen as

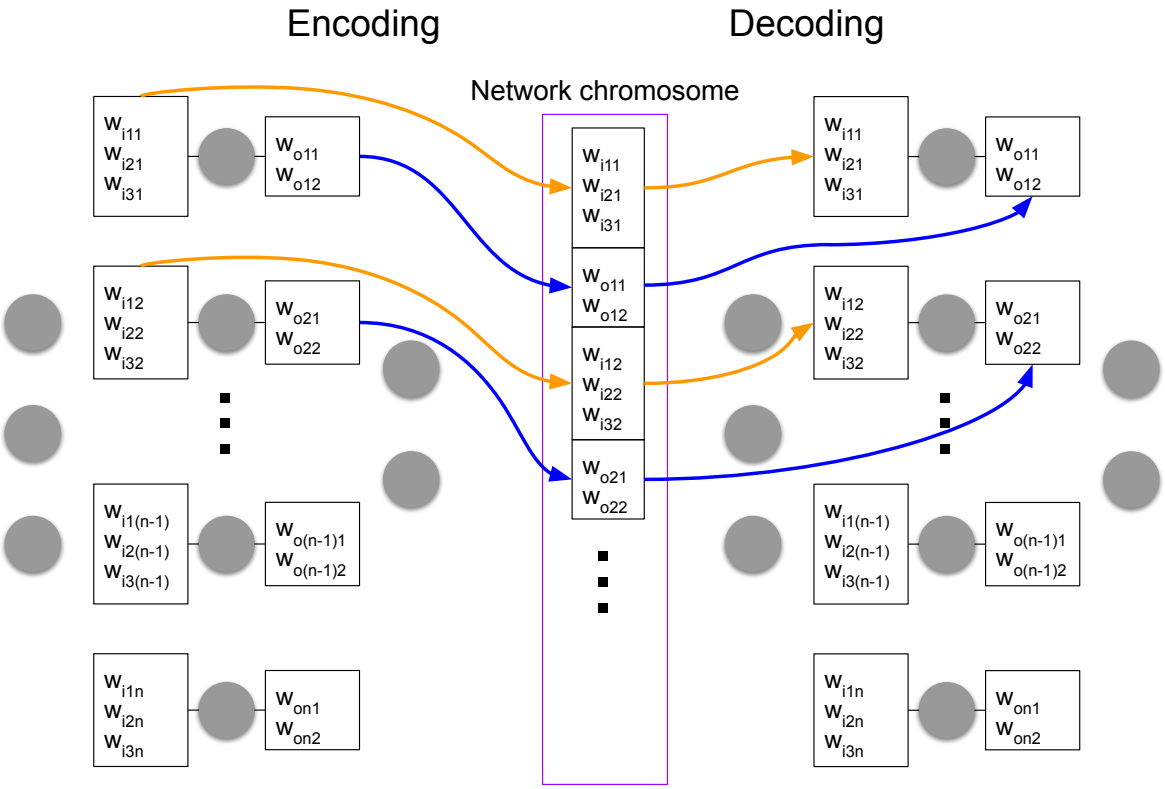

Figure 2.8: *The chromosome of each network consists of the synaptic weights connecting from the input layer to the hidden layer and from the hidden layer to the output layer.*

$$f = \frac{n_{\text{test}} N_{\max}}{S} \tag{2.27}$$

$$S = \sum_{i=1}^{n_{\text{test}}} \begin{cases} aN_i \text{ if arrived} \\ d_i N_{\max} \text{ otherwise.} \end{cases} \tag{2.28}$$

where n_{test} is the number of test rounds, $N_{\max}$ is the maximum number of steps the robot is allowed to take during each round of testing, $a = 0.1$ is a constant, N_i is the number of steps the robot has taken before arriving at the target and d_i is the final distance between the robot and the target if the robot does not arrive. In the cases where the robot reaches the target $N_i < N_{\max}$, thus the contribution aN_i to the denominator will be relatively small. If the robot does not reach the target this should give a significantly larger contribution to the denominator, and thus reducing the fitness. Therefore the contribution in those cases is set to be proportionate to $N_{\max}$. However if the robot for example collides with an obstacle close to the target this should be rewarded as better behaviour than colliding with an obstacle far away from the target, thus the the final distance d_i to the target is also factored in.

Natural selection

Selection is the process in which the best adapted individuals are favored to survive, prosper and reproduce. The selection process can be simulated in several different ways, here *tournament selection* is used. In the tournament selection two individuals are drawn at random from the population, upon which the fittest of the two is selected with probability $P_{tour} \in (0, 1)$.

Reproduction and recombination

Some biological organisms reproduce sexually, where the genome of the offspring is a mix of the genome of both parents. This process is called *recombination* and it allows for a larger diversity in the population than asexual reproduction (where the genome of the offspring is an exact copy of the single parent's genome), which in general is advantageous in evolution. Recombination in an evolutionary algorithm can be simulated by *crossover*. The principle of crossover is illustrated in Figure 2.9.

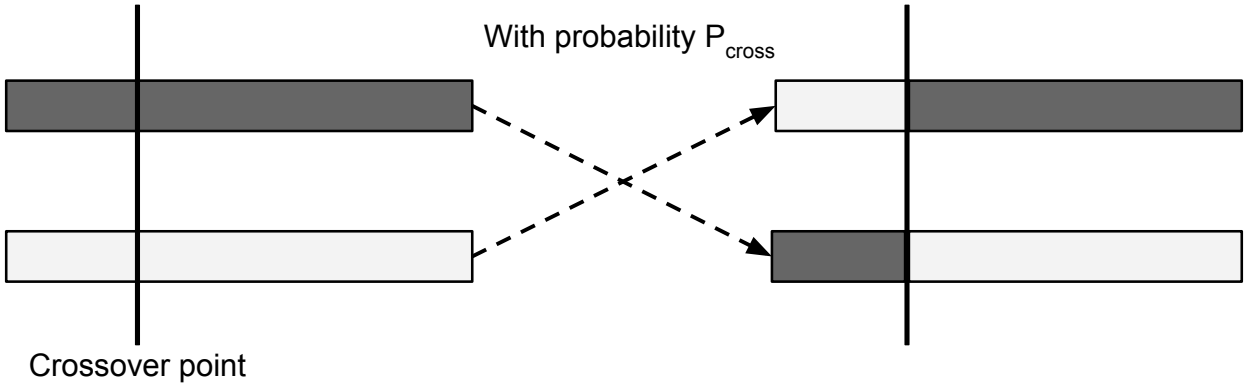

Figure 2.9: *Sexual reproduction results in the offspring's genome being a mix of the genome of the parents. In biological organisms the mixing of genomes can be very complex, in simulations it can be simplified as a simple crossing of the parents genome as illustrated in this picture. The crossover can occur at one or several locations in the genome.*

The crossover probability P_{cross} determines how likely it is that two selected individuals mate. The number of crossover points is here set to one and the location of it is randomly drawn. In reality the number and location of the crossover points can vary.

Mutation

Mutations can occur in many different ways but what they all have in common is that they result in permanent changes to the genome of an organism. In the case of the networks a mutation is represented by a gene (corresponding to a single weight in the network) changing value. Each gene is mutated with a probability of P_{mutate}.

Elitism

Elitism is a way of ensuring that the maximal fitness in the population never decreases. The fittest individual (or a few of the fittest individuals) from one generation are copied directly, without changes, into the the next generation.

3 Experiments

The spiking neural network is implemented in a simulated robot. The robot, the environment and the spiking neural network are all written in *C#*. The implementation of the robot is based off of the autonomous robot described in [16], including motor dynamics and sensors in order to make a the simulation as realistic as possible and to make a translation into a real-world robot simpler.

3.1 Robot simulation

The robot has a simple design consisting of a round body and two wheels on the left and right side of the body. The wheels are controlled by two separate DC-motors. Furthermore the robot is equipped with 3 IR-sensors, one at the front and one each at ± 1 radians angle, a compass and an odometer. Parameters for the robot and its sensors can be found in Appendix A.

3.2 Input and output signals of the network

The signals from the IR-sensors, are the input to the network controlling the robot along with information about the estimated absolute distance and (left and right) angle to the target. Each signal is scaled to be in the range $[0, 1]$. The signals given by the following

$$\text{Distance to target: } s_d = \min\left(\frac{d}{d_{\max}} + 0.3, 1\right) \tag{3.1}$$

$$\text{Left angle to target: } s_{\theta l} = \begin{cases} \frac{|\alpha|}{\pi} \text{ if target is to the left of the robot} \\ 0 \text{ otherwise} \end{cases} \tag{3.2}$$

$$\text{Right angle to target: } s_{\theta r} = \begin{cases} \frac{|\alpha|}{\pi} \text{ if target is to the right of the robot} \\ 0 \text{ otherwise} \end{cases} \tag{3.3}$$

$$\text{Left sensor: } s_{sl} = \rho_l, \tag{3.4}$$

$$\text{Front sensor: } s_{sf} = \rho_f, \tag{3.5}$$

$$\text{Right sensor: } s_{sr} = \rho_r \tag{3.6}$$

where d is the distance to the target, $d_{\max}$ is the maximum possible distance between the robot and the target, α is the angle between the robots heading and the target and ρ_l, ρ_f and ρ_r are the readings of the left, front and right sensors respectively. The sensor readings are already in the range of $[0, 1]$, thus they do not need scaling. The factor of 0.3 is added to s_d so that the signal does not approach 0 as the robot approaches the target, since this could result in the robot never actually arriving.

The output of the network is interpreted differently depending on if the network is spiking or non-spiking. In both cases the resulting signal from an output neuron is in the range $[0, 1]$ and is input to one of the DC-motors that drive the wheels of the robot. In the motors the signals are translated into applied voltages in the range $[0, V_{\max}]$. In the spiking networks the output signal of an output neuron is the average firing frequency of the neuron over some time window, scaled by the maximum possible firing frequency. In the non-spiking case the output is simply the current neuron potential.

3.3 Network structures

The network requires six input neurons: i_d for the distance to the target, $i_{\theta l} > 0$ for the left angle to the target, $i_{\theta r} > 0$ for the right angle to the target and i_{sl}, i_{sf} and i_{sr} for the readings from the left, front and right sensors respectively. The network has two output neurons, o_l and o_r, one for each motor. There is a quite intuitive way of connecting the input neurons directly to the output neurons to achieve a target approaching and collision avoiding behaviour. This connection scheme is visualized to the left in Figure 3.1. The larger the distance to the target (connected to i_d) the larger the output signals to the motors should be, in order to move

the robot. Thus the connection from i_d to the output neurons should be positive. If the target is to the left of the robot $i_{\theta l} > 0$. In order to make the robot turn towards the left the left motor signal should be weaker then the right. Thus the connection from $i_{\theta l}$ to o_r should be positive whilst the connection from $i_{\theta l}$ to o_l should be negative. By using similar reasoning the connections from the rest of the input neurons to the output neurons can be deduced. Of course the exact values of each synaptic weight are yet to be determined as this reasoning only gives the sign of each weight.

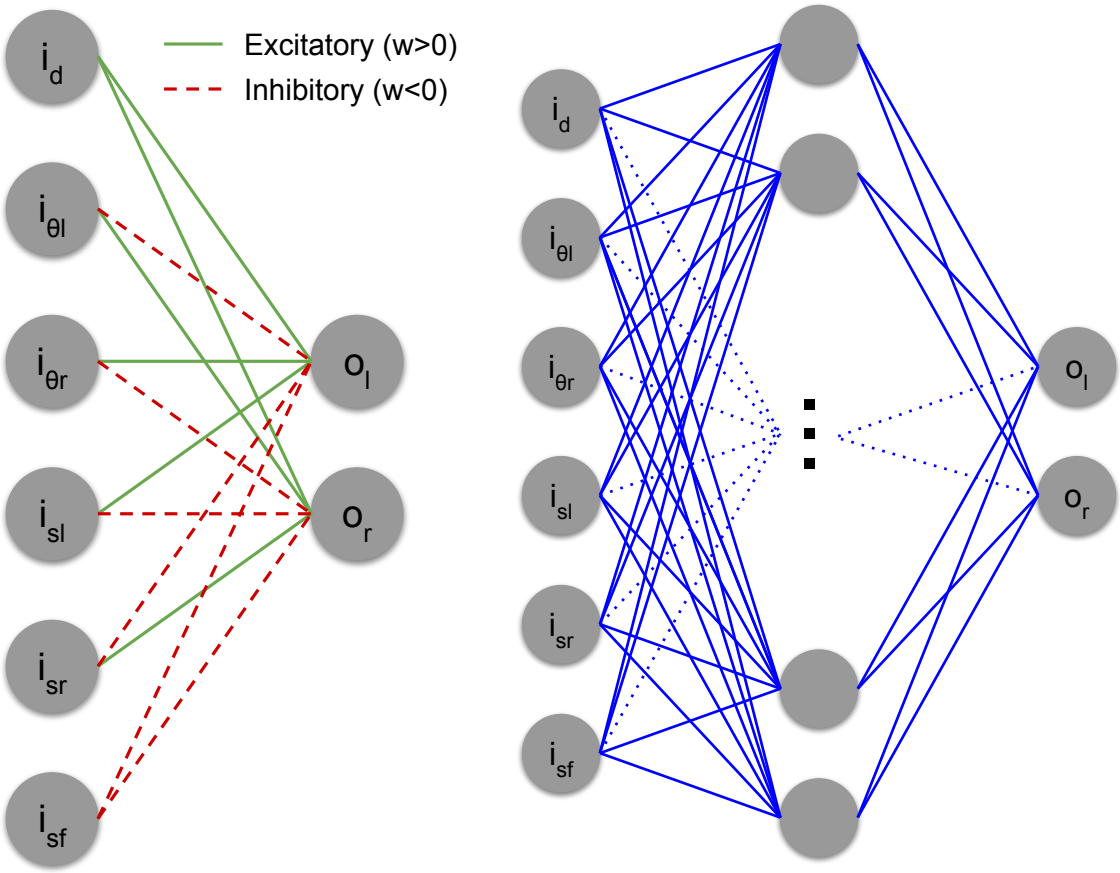

Figure 3.1: *The networks have six input neurons: i_d for the distance to the target, $i_{\theta l} > 0$ if the target is to the left of the robot and 0 otherwise, $i_{\theta r} > 0$ if the target is to the right of the robot and 0 otherwise and i_{sl}, i_{sf} and i_{sr} for the readings from the left, front and right sensors respectively. The network has two output neurons, o_l and o_r, one for each motor. In a network with no hidden layer (left) the sign of the connections from the input to the output neurons can be determined by intuition. To determine the synaptic weights in a network with a hidden layer (right) an evolutionary algorithm is used.*

When a hidden layer is added to the network (to the right in Figure 3.1), and the network is fully connected, it is perhaps less intuitive how the weight of each synaptic connection should be set. Instead an evolutionary algorithm is used in order to determine how the connections should be set.

3.4 Implementation of spiking neural networks

The spiking neural network's are modelled using LIF-neurons as described in Section 2.2.1 with parameters according to Table 3.1. To simplify computation of the membrane potential the membrane resistance and membrane time constant have been baked into the synaptic weights, w_{ij}. Since the neurons only emit action potentials in the form of Dirac δ functions at firing times t^* in the LIF-model the $RI(t)/\tau_m$ term in Equation 2.2 can be replaced by $\sum_{i=1}^{n} w_{ij}\delta(t - t_i^*)$ in the equation for a neuron j with incoming neurons $i = 1, ..., n$ and

connecting weights w_{ij}. Thus the membrane potential equation is

$$\frac{du_j}{dt} = -\frac{[u_j(t) - u_{\text{rest}}]}{\tau_m} + \sum_{i=1}^{n} w_{ij}\delta(t - t_i^*) \tag{3.7}$$

where t_i^* are firing times of neuron i. When the membrane potential reaches the threshold an action potential is emitted and the potential is reset to the reset potential u_r. The synapses are modelled as weights in the range $w \in [-1, 1]$, where synapses with negative weights correspond to inhibitory synapses and positive weights to excitatory.

Parameter	Value
Membrane time constant, τ_m	10 ms
Rest potential, u_{rest}	0 mV
Reset potential, u_r	10 mV
Threshold, ν	1 V
Refractory period, τ_r	2 ms

Table 3.1: Parameter values in the LIF-neuron model

For the input neurons, which do not have any incoming neurons, the membrane potential is modelled as

$$\frac{du}{dt} = -\frac{[u_j(t) - u_{\text{rest}}]}{\tau_m} + wS(t) \tag{3.8}$$

where $S(t) \in [0, 1]$ is the signal connected to the neuron. The neurons are updated at a frequency of 1000 Hz (in simulated time) whilst the robot is updated with a frequency of 50 Hz. This difference is due to the fact the the processing unit that corresponds to the robots brain can operate at very high frequencies (even higher than 1000 Hz if necessary) whilst the robot needs longer time to execute physical things such as moving or collecting readings from sensors. Since the network is updated 20 times between each robot execution the output signal from the network can be interpreted as the firing frequencies of the output neurons. The frequencies are scaled by the maximal possible frequency (500 Hz when taking into account the refractory period of the LIF-neurons) in order to produce output signals in the range $[0, 1]$, since the input signals of the motors should be in that range.

The excitatory synapses in the spiking neural networks are updated using the pair-based RM-STDP model described in Section 2.2.2. The parameters of the RM-STDP are listed in Table 3.2. Even though the neurons are updated at a frequency of 1000 Hz the synaptic weights are only updated at the same frequency as the robot, 50 Hz. This is due to the fact that the modulatory success signal depends on the state of the robot, thus as long as the robot's state is unaltered so is the success signal. The eligibility trace ensures that the spiking activity of the neurons in the period between synaptic updates is also taken into account.

Parameter	Value
Presynaptic temporal window, τ_{pre}	20 ms
Postsynaptic temporal window, τ_{post}	20 ms
Presynaptic weight update, A^+	0.020
Postsynaptic weight update, A^-	-0.021
Reward temporal window, τ_{pre}	200 ms

Table 3.2: Parameter values for the pair-based RM-STDP model.

The inhibitory synapses in the spiking neural networks are updated using the proposed RM-STDP model described in Section 2.2.3. The parameters of the RM-STDP are listed in Table 3.3.

Parameter	Value
Presynaptic temporal window, τ_{pre}	10 ms
Negative weight update, A^-_{inh}	-0.02
Positive weight update, A^+_{inh}	0.02
Reward temporal window, τ_{pre}	200 ms

Table 3.3: Parameter values for the proposed RM-STDP model for inhibitory synapses.

3.5 Implementation of non-spiking neural networks

A more traditional non-spiking neural network is also implemented in the robot in order to compare the performance of the spiking and non-spiking networks. Non-spiking neural networks can take many different forms. However, the components of the networks implemented here have been chosen such that the networks in as many ways as possible correspond to the spiking neural network. The synaptic weights are set to be $w \in [-1, 1]$. The neuron model is essentially the McCulloch-Pitts neuron model described in Section 2.1, but with activation function g instead of comparing the weighted sum of inputs to a threshold. The activation function of the neurons is $g(\cdot) = \max\{\tanh(\cdot), 0\}$ in order to map the inputs to $[0, 1]$.

3.6 Evolution, training and testing

A combination of the evolutionary algorithm described in Section 2.4, training and testing is used to find a spiking neural network that can navigate the robot to a target without colliding with obstacles. The synaptic connection of a non-spiking neural network are then set to be identical with the resulting spiking neural network in order to compare the advantages and disadvantages of using a spiking or non-spiking neuron models. This section further presents specifics regarding the implementation of the evolutionary algorithm as well as training and testing of the networks. Table 3.4 presents the values used for the parameters of the evolutionary algorithm. The number of neurons in the hidden layer is set to 100, which roughly corresponds to the number of neurons in the hidden layers in other projects, such as [4] and [15].

Parameter	Value
Population size, N	20
Number of hidden neurons, n_h	100
Initial synaptic weights, w_{ij}	$\{-0.5, 0.5\}$
Tournament selection parameter, P_{tour}	0.75
Crossover probability, P_{cross}	0.5
Chromosome length, l	$(6 + 2) \cdot 100 = 800$
Mutation probability, P_{mutate}	$1/l = 1/800$

Table 3.4: Parameter values for the evolutionary algorithm.

3.6.1 Initializing the population

As the first step of the evolutionary algorithm a population of N fully connected feedforward networks with n_h neurons in the hidden layer is generated, where the weights of the networks are randomly assigned one of the two values $\{-0.5, 0.5\}$ with equal probability.

3.6.2 Evaluating fitness

Each network in the population is trained and tested in order to determine their fitness (i.e ability to reach a target without colliding with any obstacles). Each network is first tested and then trained n rounds where each round consists of m different scenarios. After each round of training the network is tested again. For each round $i = 1, ..., n$ of training and testing a fitness f_i is evaluated as described in Section 2.4. The final fitness of the network is set to $\max\{f_i\}$. This is done since different networks may require different amounts of training before their best performance is reached.

Training

The networks are trained to control the robot to reach a specific target without colliding with any obstacles in its environment. The networks are trained on situations similar to but not identical with the test cases. The arena is square (6×6 m) with a square obstacle (1.5×1.5 m) in the middle, see Figure 3.2. The robot is initially placed at a random location within one of the two circles to the left or right of the obstacle with a random heading. The target is placed at a randomly chosen location from a discrete set of possible locations (green squares). The robot is then set to try to reach the target. The training is terminated when the robot reaches the target or collides with the obstacle or one of the walls. Additionally the training is terminated if the robot gets stuck (i.e is standing still for too long) or if a maximum number of time steps is reached without arrival or collision.

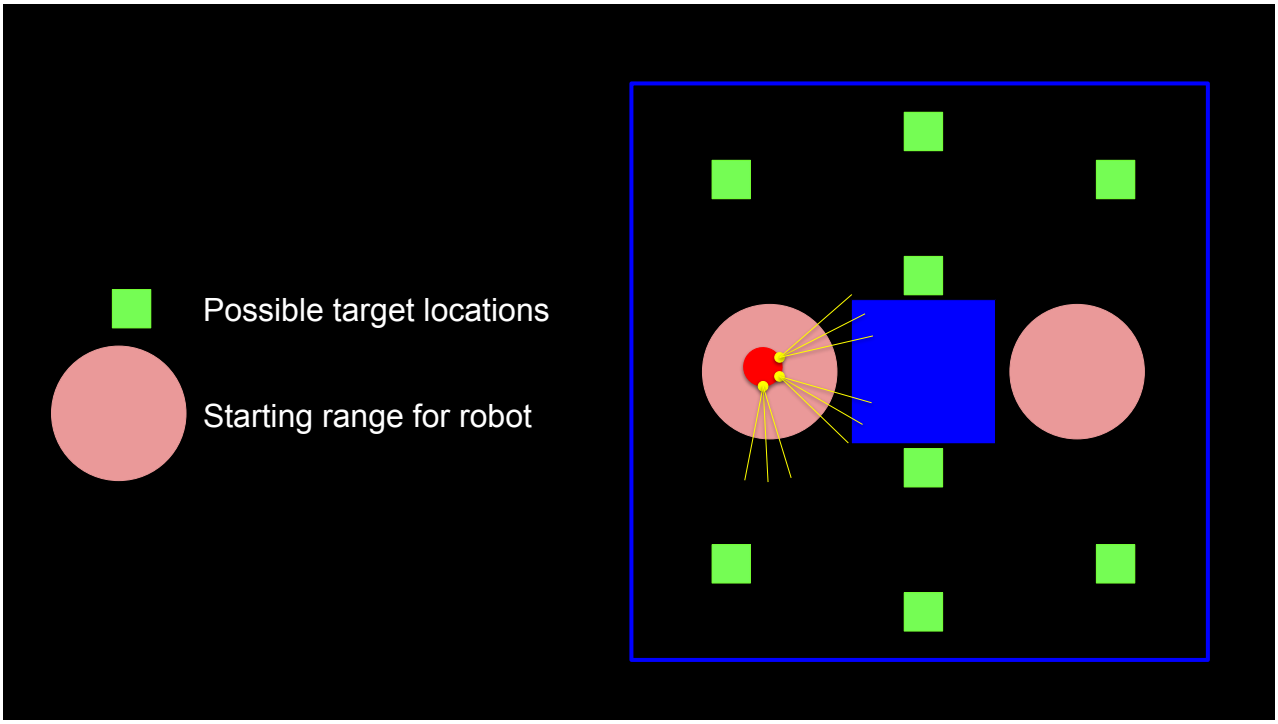

Figure 3.2: *During training the target is placed in one of the possible locations (green squares) and the robot is initially placed at a random location with a random heading in one of the two circles next to the obstacle.*

The synaptic weights of the spiking neural networks are updated during the training process using RM-STDP as described in Sections 2.2.2. All neurons in the networks are updated with the same *global* modulatory success signal, M, which depends on the performance of the network. The performance can be divided into three main categories: *distance to target*, *angle to target* and *distance to nearest obstacle*. In order to reach the target the robot must minimize the distance and angle to the target whilst simultaneously keeping the distance to the nearest obstacle sufficiently large to avoid collision. Thus M is a function of the reward parameters r_d for distance to target, r_a for angle to target and r_o for distance to nearest obstacle where the reward parameters are given by

$$r_d = \begin{cases} 1 - \frac{d}{d_v} \text{ if } d_v > d \\ 0 \text{ otherwise} \end{cases} \tag{3.9}$$

where d is the distance between the robot and the target based on the estimated position of the robot using the odometer readings, and d_v is a parameter that determines at what distance from the target the robot is deemed to be in the vicinity of the target.

$$r_a = 1 - \frac{|\theta|}{\pi/2}, \tag{3.10}$$

where $\theta \in [-2\pi, 2\pi]$ is the angle between the target and the direction of the robot.

$$r_o = r_{\text{left}} + 2r_{\text{front}} + r_{\text{right}}, \tag{3.11}$$

$$r_{\text{left}} = \begin{cases} 1 - \rho_{\text{left}}/\rho_{\max} \text{ if } \rho_{\text{left}} \geq \rho_{\text{safe}}, \\ 1 \text{ otherwise} \end{cases} \tag{3.12}$$

$$r_{\text{front}} = \begin{cases} 1 - \rho_{\text{front}}/\rho_{\max} \text{ if } \rho_{\text{front}} \geq \rho_{\text{safe}}, \\ 1 \text{ otherwise} \end{cases} \tag{3.13}$$

$$r_{\text{right}} = \begin{cases} 1 - \rho_{\text{right}}/\rho_{\max} \text{ if } \rho_{\text{right}} \geq \rho_{\text{safe}}, \\ 1 \text{ otherwise} \end{cases} \tag{3.14}$$

where ρ_{safe} is the "safe sensor reading", $\rho_{\max}$ is the maximal possible reading of the robots sensors and ρ_{left}, ρ_{front} and ρ_{right} are the readings of the left, front and right sensors respectively. The contribution from the front sensor is scaled by a factor of 2 in order to emphasise that approaching an obstacle head-on is less desirable than turning in some direction (left or right) when an obstacle is straight ahead. The modulatory success signal is given by

$$M = \begin{cases} \max\left(0, r_o - \langle r_o \rangle\right) \text{ if } d_o < d_{safe} \text{ and } d_o < d, \\ r_d - \langle r_d \rangle + r_a - \langle r_a \rangle \text{ otherwise} \end{cases} \tag{3.15}$$

d_o is the estimated distance to the nearest obstacle, d_{safe} is some "safe" distance to each obstacle and $\langle r \rangle$ denotes the average of the previous 3 values of the reward parameter. This gives the success signal a "reward minus expected reward" form. Additionally, if the robot arrives at the target the network is updated in the same way with a reward R_{arrival}, and if the robot collides with an obstacle it is updated with a penalty (negative reward) $R_{\text{collision}}$. The $\max\left(0, r_o - \langle r_o \rangle\right)$ criteria in the modulatory success signal is added with the intention that approaching an obstacle is not necessarily bad, unless the robot collides, upon which the collision penalty $R_{\text{collision}}$ is applied. The robot does on occasion have to approach an obstacle in order to reach the reward, for example if the reward is is situated close to an obstacle. That is also the reason for the $M = \max\left(0, r_o - \langle r_o \rangle\right)$ if $d_o < d$, to focus on teaching the robot to avoid moving towards obstacles, *unless* however the target is close to an obstacle where the reading from the sensors could be conflicting with the signals telling the robot to move towards the target.

Testing

After each round of training the networks are tested in order to determine their current fitness. The networks are tested on eight fixed test cases and their fitness are determined as described in Section 2.4. Figure 3.3 displays the target location and initial robot location and direction in the eight different test cases.

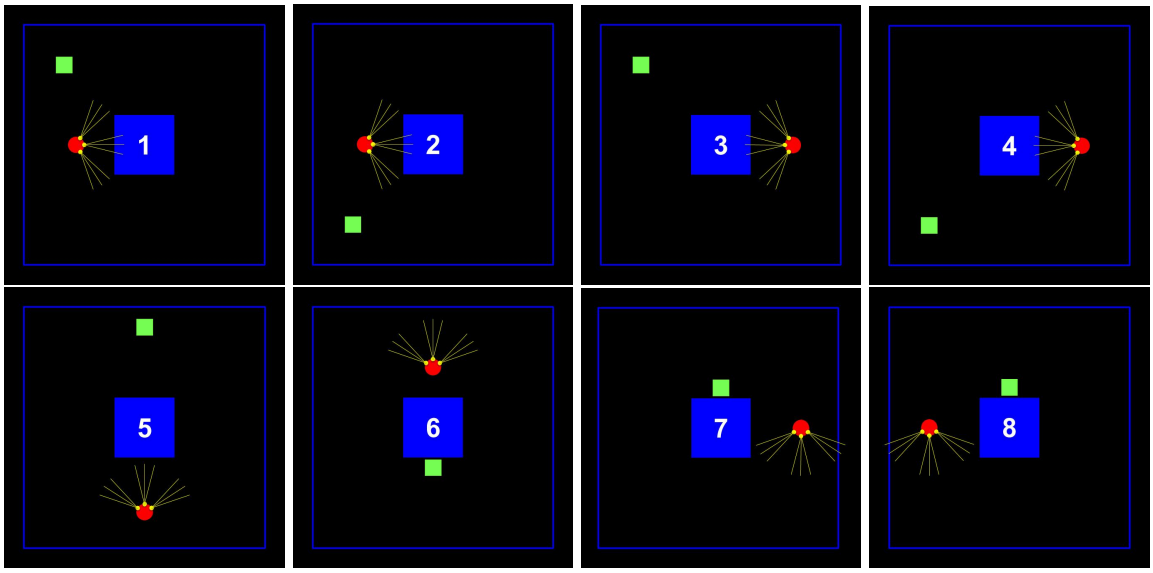

Figure 3.3: *The layouts of the eight different test cases upon which the networks fitness are determined.*

3.6.3 Generating the next generation

When each network in the population has been trained and tested and their fitness has been determined the next generation is generated. This is done in the following way:

1. The individual with the highest fitness is copied directly to the next generation (elitism)
2. One entirely new individual is randomly generated and added to the next generation in order to further increase diversity
3. Two individuals are selected from the population using tournament selection as described in Section 2.4
4. With probability P_{cross} the two selected individuals are crossed with each other as described in Section 2.4 and the offspring are copied into the next generation. Otherwise the two selected individuals reproduce asexually (without crossover).
5. The offspring from step 4 are mutated with mutation probability P_{mutate}. Since the genes can only take on of the two values $\{-0.5, 0.5\}$ a mutation is equivalent to simply flipping the sign of the gene. The mutated individuals are then added to the next generation.

Step 3-5 are repeated until the next generation has N individuals.

4 Results and discussion

4.1 Network without hidden neurons

A target navigation and collision avoidance behaviour can be achieved with a network without any hidden layers by connecting the input neurons to the output neurons according to the connection scheme to the left in Figure 3.1. By initiating the network with weights ± 0.5 the spiking neural network can make the robot reach the target in most of the test cases. Typical performance of this network is displayed in Figure 4.1. However the performance of the network seems to be sensitive to the errors in location- and direction estimation caused by the odometer and compass; the same network may arrive at the target during one test but collide with the obstacle in an identical test immediately after. The estimated direction of the robot at any given time step is equal to the actual heading of the robot with $\mathcal{N}(0, \sigma_c^2)$-noise, where $\sigma_c = 0.05$. The error in the estimated location may however accumulate over time since the estimate at each time step is based on the estimate at the previous time step. The position is updated based on the speed and angular speed of the robot with $\mathcal{N}(0, \sigma_o^2)$-noise, where $\sigma_o = 0.0002$, along with the estimated position in the previous step. Thus the error at any time step i will be a random variable with distribution $\sum_1^i \mathcal{N}(0, \sigma_o^2) = \mathcal{N}(0, i\sigma_o^2)$. In all simulations the robot is limited to a maximum of 5000 steps, after which the error in the estimation of the location of the robot will be $\mathcal{N}(0, \sigma_{5000}^2)$-distributed, where $\sigma_{5000} \approx 0.014$. In many cases the robot will take significantly fewer steps before reaching the target or colliding. In relation to to the radius of the robot, $R = 0.2$ m, and the size of the arena, 6x6 m, the expected error in the location estimation is quite small. Yet, the errors in the direction- and location estimations are significant enough to make the networks perform inconsistently between different runs.

What is perhaps noteworthy is that the target never arrives at target number 5. Since the inputs are symmetrically connected to the outputs of the network the situation where the robot is positioned directly towards the target, but with an obstacle between them, results in a deadlock and the robot ends up standing still. The slight asymmetry caused by the errors in the odometer and compass do not seem to be sufficient to budge the network out of the deadlock. In test case 6, when the robot is facing away from the target, any small error in the estimation of the heading of the robot will result in a signal close to 1 in either the left or right angle input neuron. This causes the robot to start turning and not end up in a deadlock as in test case 5.

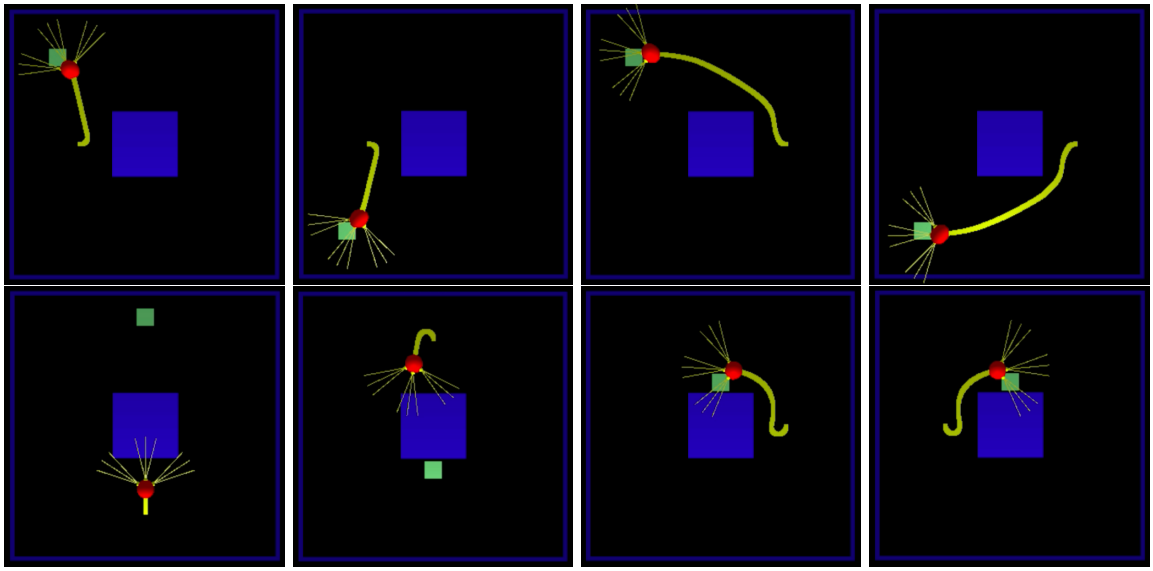

Figure 4.1: *A network connected as to the left in Figure 3.1 with weights set to* ± 0.5 *arrives in most of the test cases, however the robot never arrives in test case 5 and sometimes gets stuck in a deadlock in test case 6.*

By training as described in section 3.6.2 the performance of the network can be improved, although the exact performance depends on the scenarios that the network is trained on, which varies from attempt to attempt since the scenarios are randomly generated. Figure 4.2 presents the average number of arrivals μ after different amounts of training along with the standard deviation σ. The averages are computed by training on 20

randomly generated scenarios at a time and and then testing 10 times after each round of training. This is repeated until the robot has been trained on a total of 200 different scenarios. The process is then repeated 9 times with different seeds in the random number generator. To the right in Figure 4.2 the average number of arrivals are plotted for the different random number generator seeds.

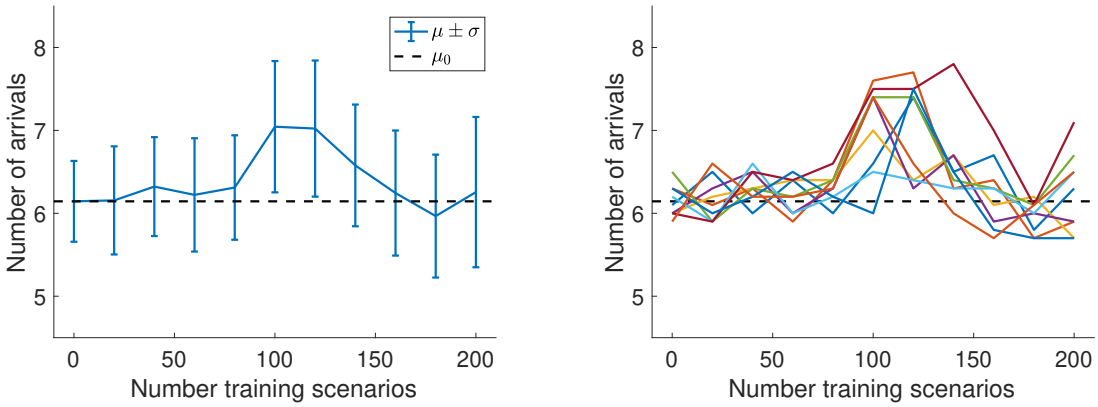

Figure 4.2: *Average number of arrivals μ after training with standard deviation σ plotted against the number of rounds of training. The average number of arrivals when the network has no training is indicated as μ_0.*

As expected the exact performance varies, since the networks are trained on different scenarios. Yet it seems that, in most cases, at approximately 100-120 training scenarios the network tends to reach its peak performance. Figure 4.3 displays how the network performs on the eight test cases after training as described in Section 3.6.2 on 120 different scenarios the robot can arrive at the target in all eight test cases.

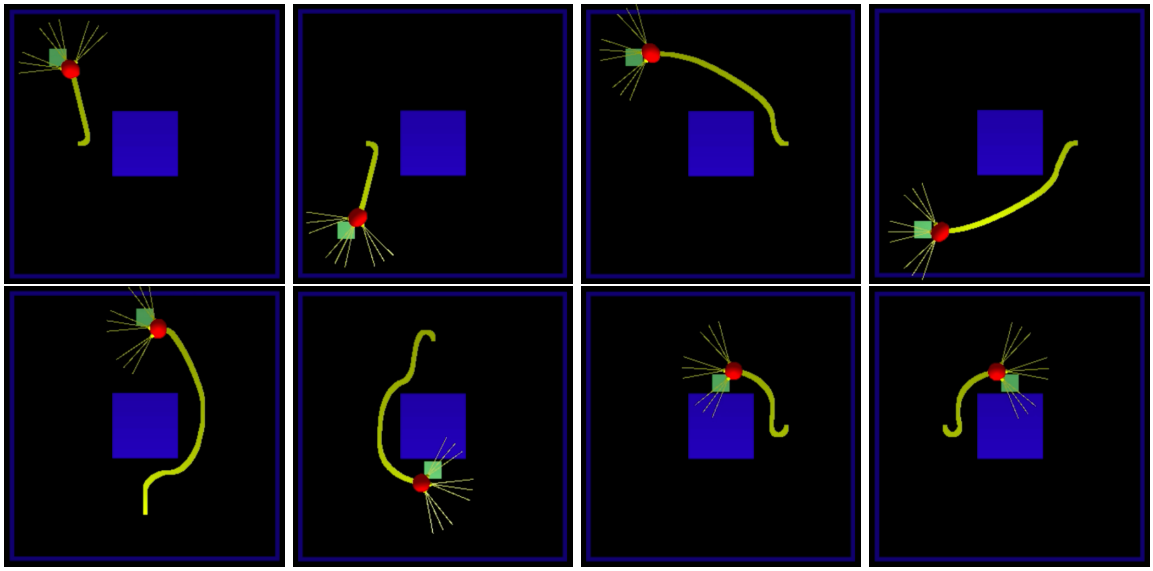

Figure 4.3: *After training the network to the left in Figure 3.1 with weights set to ± 0.5 as described in Section 3.6.2 on 120 different scenarios the robot can arrive at the target in all eight test cases.*

It can be noted that the average number of arrivals increases with training, up to a certain point, meanwhile the standard deviation does not decrease. When testing the network several times after training different runs still yield different results. This indicates that the performance of the network still is sensitive to the errors in the location and direction estimations, even after training. After approximately 120 rounds of training the performance of the network decreases steadily. This is due to a form of over training, further discussed in later on in Section 4.4.

4.1.1 Impact of the proposed STDP for inhibitory synapses

In order to determine whether the proposed STDP for inhibitory synapses, abbreviated here as I-STDP, actually improves learning the network is trained both with the original STDP (Section 2.2.2, for both excitatory and inhibitory synapses, and with I-STDP for inhibitory synapses. The network is trained on 20 randomly generated scenarios at a time and and then tested 10 times after each round of training. The network is trained with the two different methods on identical training scenarios. The average number of arrivals after different amounts of training are presented in Figure 4.4, where different figures represent different sets of training scenarios. In all cases examined I-STDP yields a maximum that is higher than (or equal to) the corresponding maximum for the original STDP. The maximum also is often reached earlier when using I-STDP. This indicates that the proposed I-STDP in fact does improves the learning.

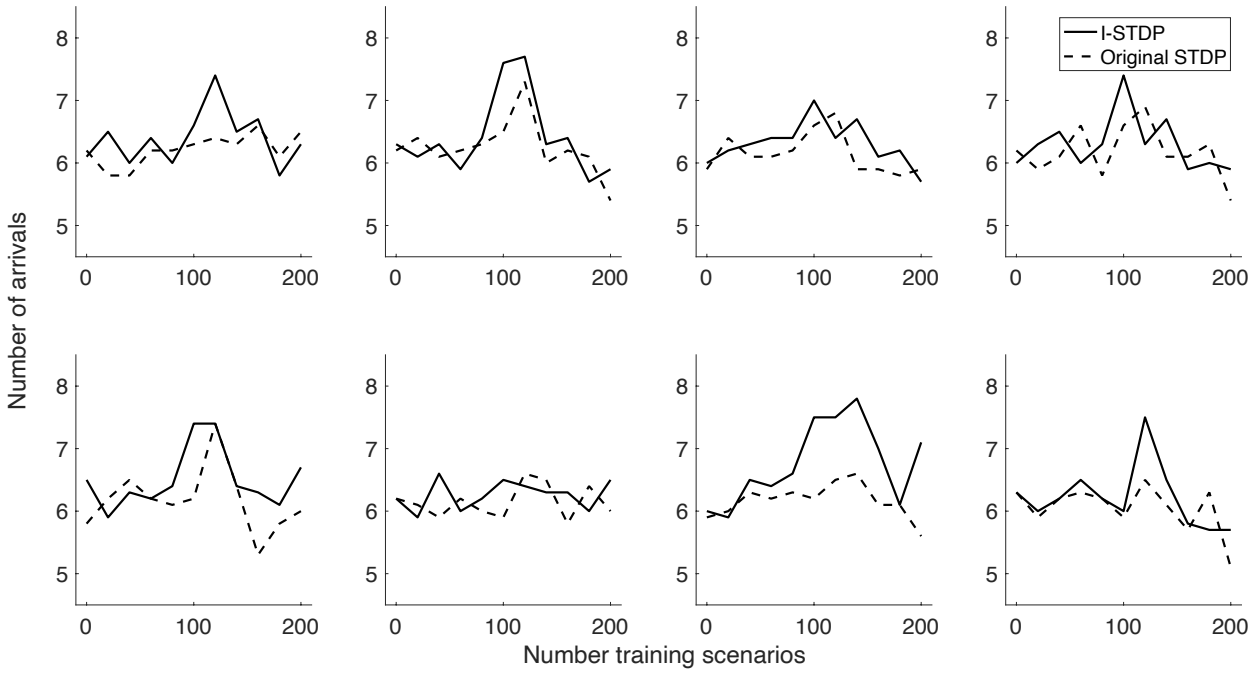

Figure 4.4: *Average number of arrivals after training with the proposed STDP for inhibitory synapses, I-STDP, and after training with the original STDP.*

4.2 Evolved network with hidden layer of neurons

The performance achieved by the two-layered network shows sensitivity to errors position and location estimation, which indicates that such networks do not allow the robot to take careful enough precautions around obstacles. Perhaps adding a hidden layer with a sufficient amount of neurons could allow for information to be encoded such that the network can make the robot both more efficiently avoid the obstacles and maintain the target-approaching behaviour. This section presents the results from the evolutionary algorithm described in Section 2.4. The initial population of networks displays a wide range of random behaviour. After a few generations of the evolutionary algorithm some form of target approaching behaviour starts to appear in the population. Over time the goal approaching and obstacle avoidance becomes more and more robust, giving the individuals in the population higher fitness. Although the networks get better at reaching the target in the test cases they often display a form of lopsidedness during evolution, always turning in the same direction when avoiding an obstacle or approaching the target. An example of this is displayed in Figure 4.5. This lopsidedness does not necessarily prevent the robot from reaching the target, but it may result in the robot taking unnecessary detours. Furthermore, in a more advanced layout of the arena, a lopsidedness in the ability to avoid obstacles may cause more trouble.

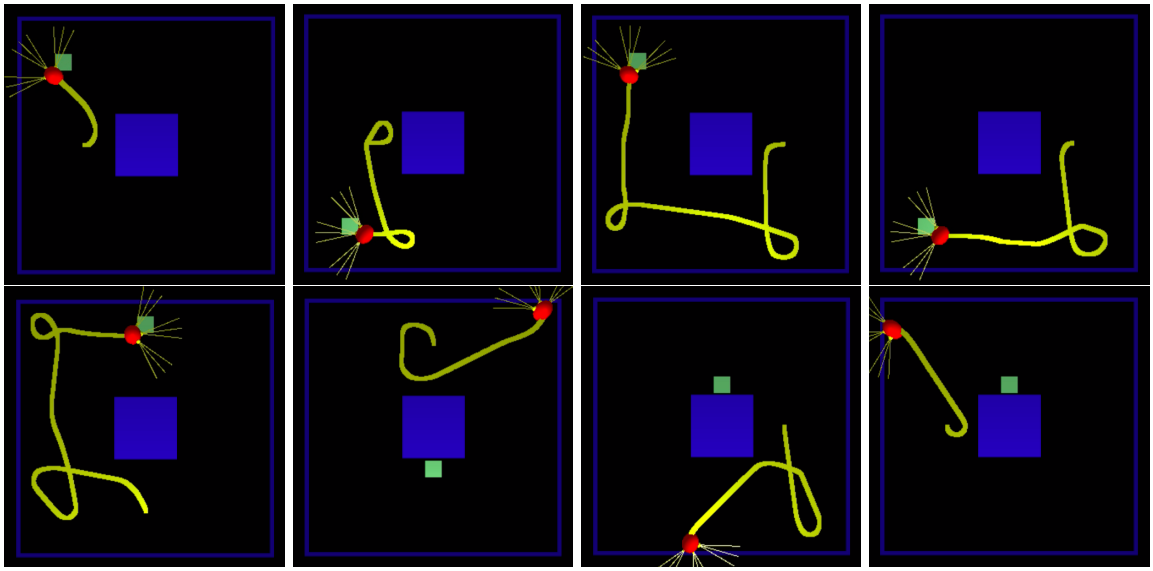

Figure 4.5: *During evolution networks may display some form of lopsidedness. Here the network makes the robot turn left when avoiding an obstacle which leads to the robot taking a longer route to the target.*

By letting the population of networks evolve further the lopsidedness decreases since the fitness measure increases when the number of steps before arriving at the target decreases. A network that takes the shortest possible way to the target will have a higher fitness than a lopsided network that takes detours before arriving. However a common problem with evolutionary algorithms is *premature convergence*, that the algorithm converges to a fitness below the actual maximum. Figure 4.6 shows how the maximum fitness of the population changes over 30 generations for two separate runs of the algorithm. In one of the rounds (red dashed line) the fitness does not not change much over time, a typical sign of premature convergence. The choice of parameter values in the evolutionary algorithm impact the risk of premature convergence, however determining the optimal parameter values may be difficult. Another way to potentially work around premature convergence is to rerun the algorithm several times. Due to the stochasticity of the algorithm different runs are likely to yield different results and perhaps one of several runs could converge to the actual optimum. Furthermore there are no guarantees that a network in the defined search space (100 hidden neurons, initiated with weights ± 0.5 and trained as previously described) actually can achieve the desired behaviour.

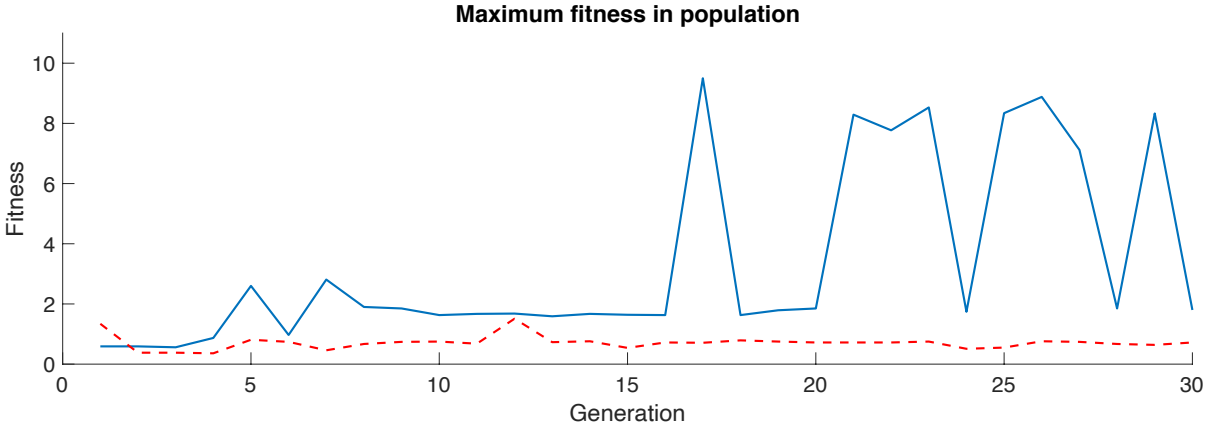

Figure 4.6: *The maximum fitness of the population over 30 generations for two separate runs of the algorithm. In one case (blue full line) the fitness increase initially, but becomes quite volatile towards later generations. This is due to the networks being sensitive to errors in the location and heading estimations and that the networks are being trained on different scenarios in different generations, which naturally can result in varying performance. In the other case (red dashed line) the fitness does not change significantly, a typical sign of premature convergence.*

In the other round of evolution displayed in Figure 4.6 (blue full line) the fitness does increase from the initial value, but becomes quite volatile towards later generations. Even if the individual with the highest fitness in one generation is copied directly to the next generation (elitism), that network may not achieve the same fitness two generations in a row. This is again due to the networks being sensitive to errors in the location and heading estimations and that the networks are being trained on different scenarios in different generations, which naturally can result in varying performance. This volatility could perhaps be remedied to some extent by testing the networks several times after each round of training and computing some form of average fitness over several tests. Such a fitness measure might benefit networks with more consistent performance and less sensitivity to the estimation errors.

The performance of the best network achieved from 30 generations of evolution is displayed in Figure 4.7. The network reaches its best performance after approximately 40 rounds of training, which is significantly less than for the network with no hidden layer. However there is still a slight lopsidedness to the network and the sensitivity to the errors in the estimation of the location and heading of the robot remains.

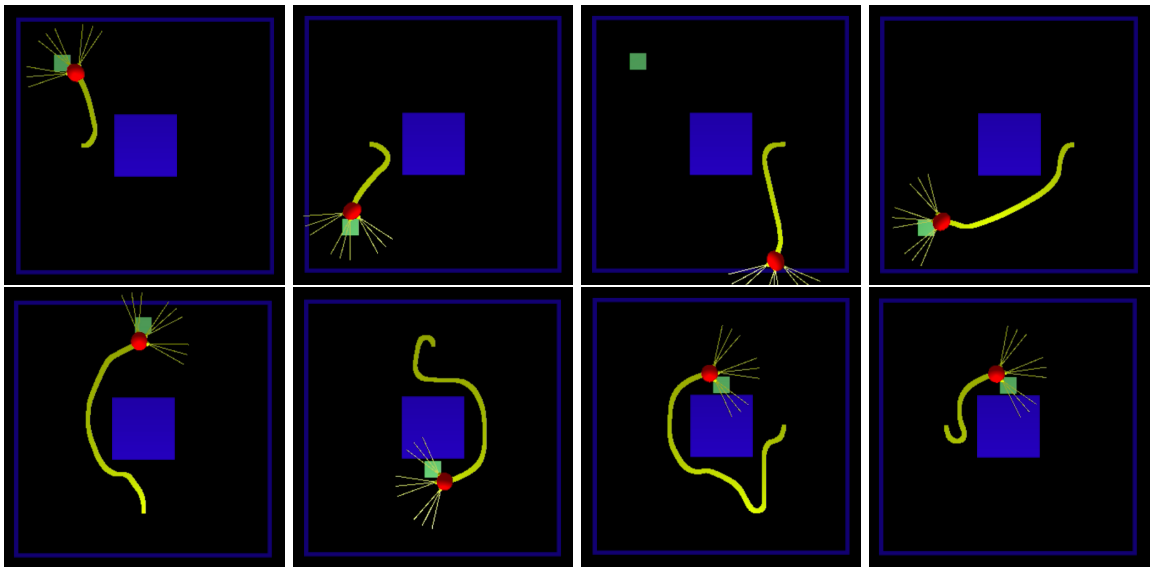

Figure 4.7: *The best network from the 30 generations of evolution. The network reaches its best performance after approximately 40 rounds of training, however there is still a slight lopsidedness to the network and the sensitivity to the errors in the estimation of the location and heading of the robot remains.*

Other than getting the robot simulations up and running and implementing the spiking and non-spiking neural networks a significant portion of the work in this project has gone to determining the various reward parameters in the STDP as well as defining several aspects of the evolutionary algorithm. Unfortunately, as the algorithm turned out to be quite time consuming to run on the available hardware (and due to some unfortunate errors and mistakes), this process has taken quite some time. Ideally the algorithm would have been run several more times, both with the parameters in Table 3.4, and with other parameters in order to better explore the potential of the algorithm. In Section 5.2 suggestions are made as to how one could proceed with further work based on this project as well as other questions that would be interesting to explore.

4.3 Spiking vs. non-spiking neural networks

Both the non-spiking network with no hidden layer and the same connection scheme as in Section 4.1 and the non-spiking version of the evolved network in Section 4.2 perform better than their spiking counterparts. The non-spiking neural networks consistently reach all 8 targets in the test-cases, even without training. If one interprets the value of a non-spiking neuron as the spiking rate over some time window this would mean that the spiking rate could take any real value in the range $[0, 1]$, when in fact, due to the refractory period of a neuron and the limited time window in which the spiking rate is measured, this is not quite true. Even if certain information may be lost when the exact spiking times of the neurons are not considered in the

spiking rate-interpretation in non-spiking neural networks it would seem that it in fact is the firing rate and not the exact timing of neuron spikes that determines the output of the network. The higher resolution of the spiking rate in the non-spiking neurons gives the network a better performance. This is likely also why the symmetrically connected non-spiking neural network still manages to reach the target in test-arena 5; the small asymmetries from the errors in the odometers and compass are enough to offset the network and allow the robot to navigate around the obstacle, as a result of the higher resolution of spiking rate in the neurons. It is possible that translating the output of the networks into scaled spike frequencies leads to some of the information encoded in the exact timing of spikes being lost. Furthermore the simulations here are run at exact frequencies where the neurons are updated every 1 ms and the output of the network computed every 20 ms. In a biological network these processes likely do not occur on exact frequencies. Perhaps simulating the translation from spikes to actions in a fashion more similar to that of a motor neuron would yield better performance for the spiking neural networks.

Even if the test arenas have been selected to test the networks' obstacle avoidance in different situations they do not fully represent all possible situations. To get a better idea of how the spiking and non-spiking networks perform, and to ensure that the results from the test-arenas are not biased by the narrow selection of cases, the networks are also tested on 1000 randomly generated arena layouts as in figure 3.2. The number of arrivals are presented in table 4.1. Both the synaptic connections as described to the left in Figure 3.1 with weights ± 0.5 and the connections resulting from training are tested. The network without a hidden layer is trained on 120 different scenarios and the network with a hidden layer of 100 neurons is trained on 40 different scenarios.

Type of network	Without training	With training
Spiking, no hidden	821	842
Non-spiking, no hidden	923	914
Spiking, 100 hidden	758	784
Non-spiking, 100 hidden	762	916

Table 4.1: Number of arrivals when testing on 1000 random arena layouts as in figure 3.2. The networks are tested before training, with weights set to ± 0.5, and with the weights resulting from training as described in Section 3.6.2. The network with no hidden layer is trained on 120 different scenarios and the network with a hidden layer of 100 neurons is trained on 40 different scenarios.

The non-spiking networks perform better also on the randomly generated test cases, both with the trained and untrained weights. The performance of the spiking neural networks increase somewhat after training, but they still arrive in significantly fewer cases than the corresponding non-spiking networks. The network with no hidden neurons seems to perform better than the evolved network with a hidden layer. This could to some extent be due to the fitness in the evolutionary algorithm being determined by the eight test cases, thus there is no guarantee that the evolved network performs as well on other arena layouts. Interestingly, in the network with no hidden neurons, the performance of the non-spiking network is slightly worse with the trained weights than the with all weights set to ± 0.5, although the performance is still better than the corresponding spiking network. On the other hand the non-spiking network with a hidden layer performs significantly better with the trained weights than with the weights set to ± 0.5, arriving in approximately the same number of cases as the non-spiking networks without hidden neurons.

4.4 Training and overtraining

The reward used in training the networks, as described in Section 3.6.2, is based on the position of the robot in relation to the target and obstacles. The reward is also compared to the reward in the previous few time steps, giving it a *reward minus expected reward*-form. This means that firing patterns that make the robot approach the target and avoid the obstacles are enhanced whenever the robot exhibits those behaviours. The idea is that by training the network on a sufficient number of diverse arena layouts the synapses of the network should be tuned such that the network tends towards the the target while avoiding obstacles in the most efficient manner.

For each specific task there exists some optimal synaptic connection which gives a network the best possible performance. However, the reward-scheme implemented here does not take that into account. If a network were to have the ideal values of each synapse, training the network with this reward-scheme would lead to

certain synaptic connections being strengthened and others weakened, thus moving the network away from the ideal synaptic connection and reducing the performance of the network. This is a form of *over training*; for each network there exists an ideal amount of training (on an ideal selection of training scenarios) that gives that specific network the best possible performance under the applied reward-scheme. However, determining the ideal set of training scenarios is a not a simple task. Therefore the approach in this project has been to randomly generate the training scenarios, as described in section 3.6.2, the idea being that if the network is set to train on a sufficient amount of randomly generated scenarios most relevant situations should be covered. Additionally the rate of learning is quite low, due to the reward minus expected reward approach, thus reducing the risk of a network prematurely adapting to whatever situations it is presented with early on during training and allowing each network to train on a larger amount of training scenarios. This is also why the performance on the selected test arenas is tested several times during training in the evolutionary algorithm, as different networks may reach their optimal performance after different amounts of training.

Of course using a different reward-scheme could remedy some of these problems. For example one could calculate the ideal trajectory from the starting position of the robot to the target and then base the reward on the robot's deviation from that trajectory. A network that already makes the robot follow the ideal trajectory could then remain unaltered during training. However calculating the ideal trajectory could be difficult and this approach may also unnecessarily punish networks that for example find alternate ways of avoiding obstacles, causing them to deviate from the ideal trajectory but perhaps still arriving at the target. Another approach could be to somehow base the reward on the performance in the current round of training compared to the performance in previous rounds. Again, it might not be entirely trivial to do this kind of comparison, as different rounds of training require different actions and numbers of steps. One could also scale the reward with a "learning rate" which decays over time as the training progresses, perhaps by reducing the learning rate each time the robot arrives at a target during training.

4.5 More advanced arena layouts

The networks are trained and tested on a very simple arena layout, see Figure 3.2. Although the layout is very simple, with a single square obstacle in a square arena, some form of target approaching and obstacle avoiding behaviour is necessary in order for the network to make the robot reach the target. For advanced layouts the networks may need more information than just the distance and angle to target and the distance to nearest obstacle in order to navigate in any efficient manner. However, the six inputs used in this project and the training on the simple design may still be sufficient to get the robot to navigate to the target on slightly more complex layouts. The networks are tested on two additional arenas: one with four square obstacles and one with a well-like obstacle. Figure 4.8 shows how the network with no hidden layer, trained on the original simple arena, performs on the arena with four obstacles. Figure 4.9 shows how the best performing evolved network, also trained on the simple arena, performs on the arena with four obstacles. The exact performances of the networks again vary between runs. Both networks arrive at some targets but collide or get stuck in a deadlock in other situations. It seems that the network with no hidden layer has a higher tendency of getting stuck in a deadlock, whilst the network with the hidden layer will travel a longer distance and thus get closer to the target before colliding. It is possible that both networks could perform significantly better if they were trained on this arena layout.

Figure 4.10 shows how the network with no hidden layer, trained on the original simple arena, performs on the arena with a well-like obstacle. Figure 4.9 shows how the best performing evolved network, also trained on the simple arena, performs on the arena with a well-like obstacle. Also here the exact performance of the networks again vary between runs, but here the network with a hidden layer tends to reach the targets more often than the network with no hidden layer. This indicates that the hidden layer might make it possible for the network to navigate better through the more advanced layouts. It is also possible that his difference in performance could be reduced if the networks are trained on this specific layout.

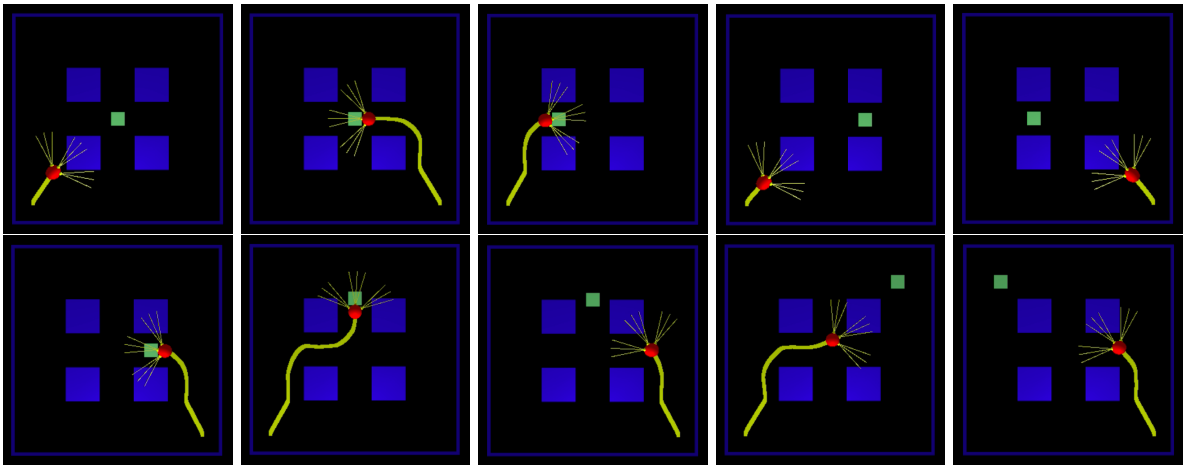

Figure 4.8: *The best performing network with no hidden layer tested on an arena with four square obstacles.*

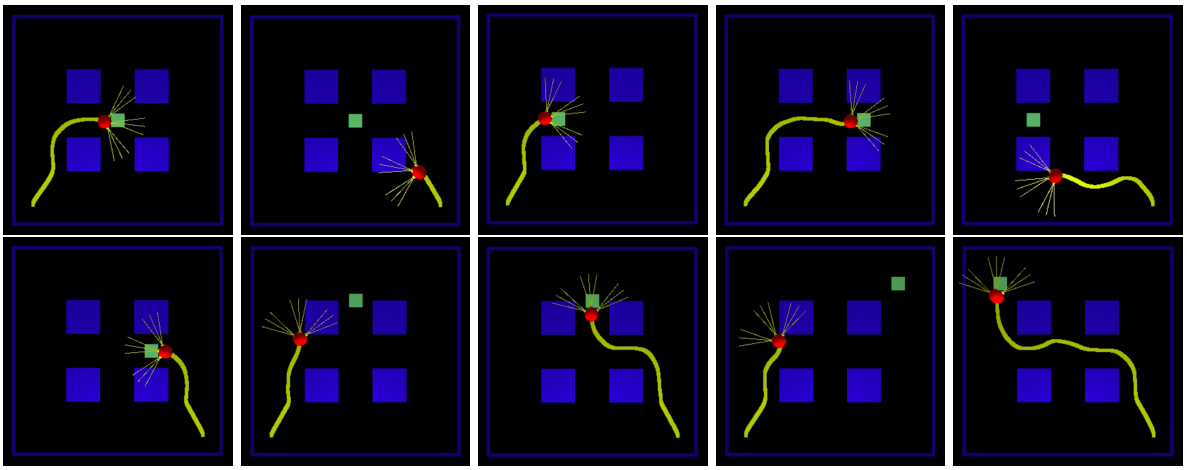

Figure 4.9: *The best performing evolved network with a hidden layer of 100 neurons tested on an arena with four square obstacles.*

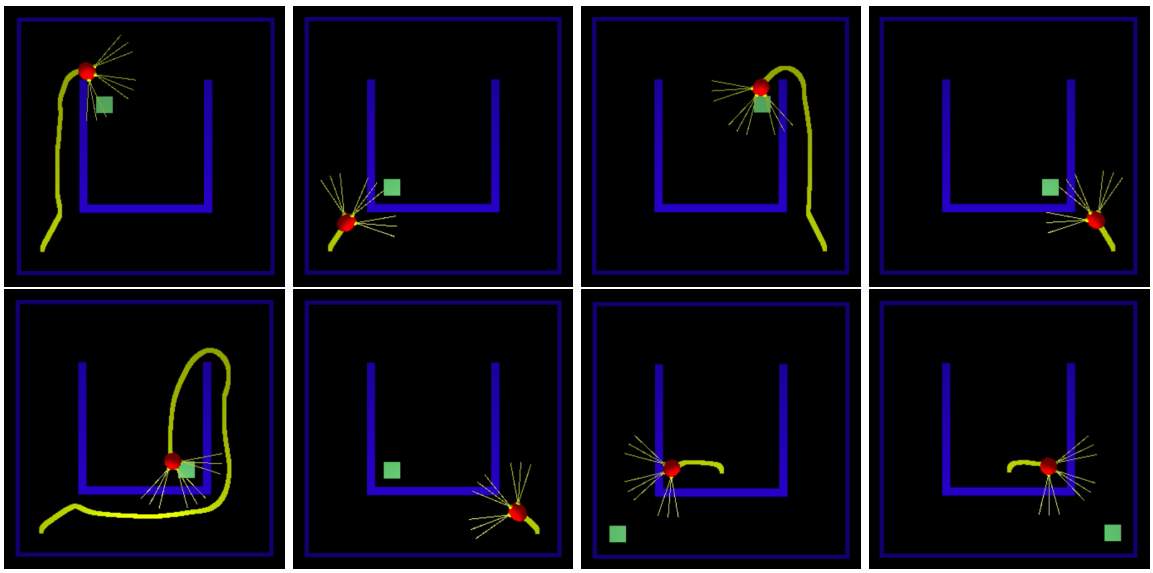

Figure 4.10: *The best performing network with no hidden layer tested on an arena with a well-like obstacle.*

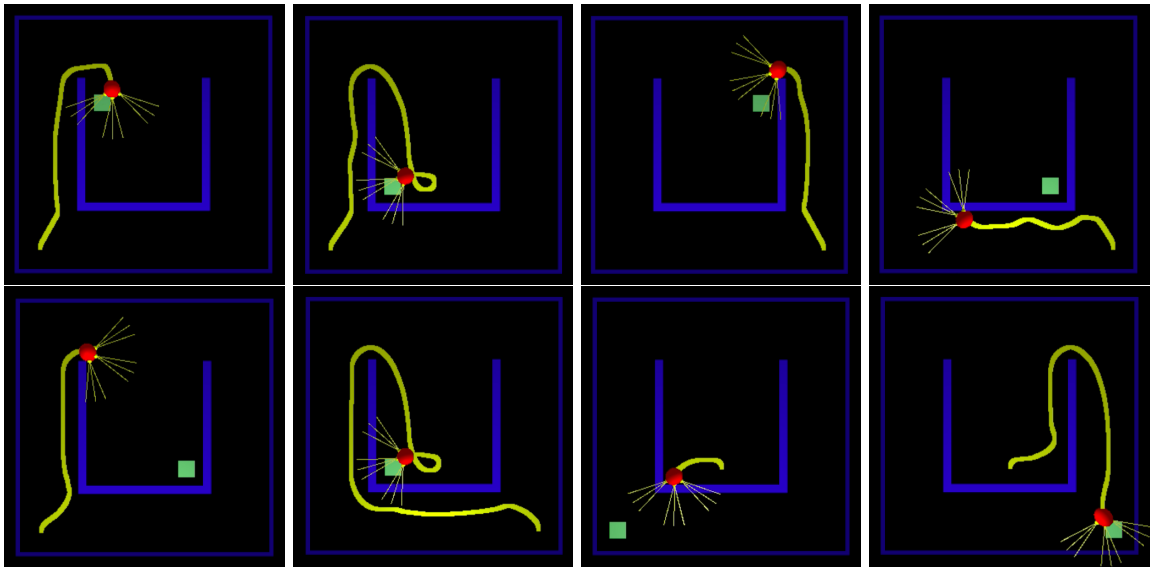

Figure 4.11: *The best performing evolved network with a hidden layer of 100 neurons tested on an arena with a well-like obstacle.*

5 Conclusions and further work

5.1 Summary of results

A decent target navigation and collision avoidance behaviour can be achieved with a network without any hidden layers. The performance can be further improved by training with RM-STDP. The proposed modified STDP for inhibitory synapses yields even better performance in the networks and often after fewer rounds of training. Maximum performance is reached after training on approximately 100 scenarios. However the two-layered network displays a sensitivity to errors in the estimated position and heading. A corresponding non-spiking neural network displays better resilience against these errors and a better performance on the target navigation and collision avoidance tasks. When tested on 1000 different arena layouts the spiking neural network arrived at the target in 842 cases, and the non-spiking network arrived in 914 cases. This indicates that the advantages of the spike-rate interpretation of non-spiking neurons outweigh the expected advantages of encoding information in the specific timing of spikes in spiking neural networks, at least in this simple network structure and the task at hand.

The proposed evolutionary algorithm shows potential of working as a tool for determining the synaptic connectivity of spiking neural networks. In one run of the algorithm, with networks with a hidden layer consisting of 100 hidden neurons, a network with the ability to arrive at most of the eight test cases was evolved in less then 20 generations. However the evolved network exhibits a sensitivity to errors in the estimation of the location and heading of the robot, just as the network with no hidden layer does. The difference between the spiking and non-spiking networks is even more significant with the addition of the hidden layer. The evolved spiking neural network arrived at the target in 784 of 1000 test cases and the corresponding non-spiking network arrived in 916 cases. Furthermore the algorithm is prone to premature convergence. This project only explored a first rather simple implementation of the algorithm, further work is necessary in order to determine the true potential of this approach. Furthermore the algorithm could be altered in many ways, for example allowing for more advanced network structures. By altering the fitness measure and the reward parameters of the RM-STDP it is possible that the algorithm can be used to determine network structures for completely different tasks than the targeted navigation and collision avoidance of this project.

5.2 Suggestions for further work

There are many ways to alter the methods implemented in this project which potentially could yield different, and perhaps better, results. Proposed ways of altering the reward scheme have been discussed in the previous section, Section 4.4. The evolutionary algorithm can be tweaked in many ways; the probabilities of mutation and crossover, the tournament selection parameter and the number of individuals in the population are all examples of parameters that can be altered and which could change the outcome of the algorithm. However these changes will likely mainly impact the number of generations needed before a network of similar performance is found. Altering the fitness measure could perhaps have a more significant impact on the results of the evolutionary algorithm. One major improvement that could be implemented in order to make the algorithm run faster is to perform the evolution and training without animating the robot and the arena. The animation process is the foremost reason as to why the algorithm is so time consuming, it could run significantly faster if all the computations were run without animating each time step.

The perhaps most interesting aspects to further investigate would be the number of neurons in the hidden layer, the ratio of inhibitory versus excitatory synapses, the range of the synaptic weights and other network structures than strict feedforward networks. In this project the number of neurons in the hidden layer is set to 100, which roughly corresponds to the number of neurons in the hidden layers in other projects. Some initial tests were done with a smaller hidden layers (10, 20 and 60 neurons), but the preliminary results indicated of quite poor performance. However it is possible that similar (or better performance) could be achieved also with fewer neurons if the algorithm is run with the right parameters, or perhaps by initiating the network weights at a larger range of values. It would be interesting to see how the performance scales with the number of neurons and how few neurons are sufficient in the hidden layer for the network to maintain the ability to navigate to the target without collisions. Furthermore there might be some optimal number of neurons. It would be quite straight forward to alter the algorithm such that the individuals of a population can have

different number of neurons in their hidden layers, thus allowing the number of neurons of the hidden layer to also be determined by the evolutionary algorithm. The number of neurons required likely depends on the values of the synaptic weights, by initiating the neurons at values other then ± 0.5 a different performance could be achieved. One could for example choose a range of discrete values to initiate the synaptic weights with or assign them real values in the range $[w_{\min}, w_{\max}]$. This, of course, increases the search space of the algorithm and thus might increase the number of generations required. In biological neural networks the ratio of inhibitory versus excitatory synapses is in general is not found to be 50/50, in fact the number of inhibitory synapses tends to be outnumbered by the number of excitatory ones. The ratio 50/50 was here chosen because that was the ratio in the connection without any hidden layer (left in Figure 3.1), but it could very well be that a different ratio is better for multilayered networks. Finally, the perhaps most interesting feature to explore, would be other network structures than strict feedforward. By modifying how the chromosomes are generated from the networks one could also encode how each neuron connects to every other neuron, thus allowing the the connectivity of the network to be determined by the evolutionary algorithm. In biological neural networks the connections between neurons are highly complex and generally do not have a strict feedforward structure. It would be interesting to see if better performance could be achieved using more intricate connection schemes. Perhaps fewer neurons would be required, yielding a more energy efficient network.

In this project, and in most of the papers that have inspired this work, the spiking neural networks have been feedforward. Energy efficiency is one of the great enigmas of biological neural networks; the human brain has an immense computational power but only requires approximately 20 Watts [3]. Arguments have been made that hardware implementations of spiking neural networks perhaps could lead to lower energy consumption and faster execution than corresponding implementations of traditional neural networks. Making the execution of the networks run efficiently has not been a main focus of this project, thus no conclusions can be drawn from these simulations about the computational power required of the spiking neural networks versus the non-spiking networks.

References

[1] F. Alnajjar and K. Murase. "Self-Organization of Spiking Neural Network Generating Autonomous Behavior in a Miniature Mobile Robot". Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 255–260. ISBN: 978-3-540-29344-6.

[2] R. Batllori et al. Evolving spiking neural networks for robot control. *Procedia Computer Science* **6** (2011), 329–334. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2011.08.060. URL: http://www.sciencedirect.com/science/article/pii/S1877050911005254.

[3] Z. Bing et al. A Survey of Robotics Control Based on Learning-Inspired Spiking Neural Networks. *Front. Neurorobot.* **12** (2018), 35. ISSN: 1662-5218. DOI: 10.3389/fnbot.2018.00035. URL: https://www.frontiersin.org/article/10.3389/fnbot.2018.00035.

[4] Z. Bing et al. Supervised Learning in SNN via Reward-Modulated Spike-Timing-Dependent Plasticity for a Target Reaching Vehicle. *Front. Neurorobot.* **13** (2019), 18. ISSN: 1662-5218. DOI: 10.3389/fnbot.2019.00018. URL: https://www.frontiersin.org/article/10.3389/fnbot.2019.00018.

[5] D. Floreano and C. Mattiussi. "Evolution of Spiking Neural Controllers for Autonomous Vision-Based Robots". Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 38–61. ISBN: 978-3-540-45502-8.

[6] W. Gerstner et al. Eligibility Traces and Plasticity on Behavioral Time Scales: Experimental Support of NeoHebbian Three-Factor Learning Rules. *Frontiers in Neural Circuits* **12** (July 2018). ISSN: 1662-5110. DOI: 10.3389/fncir.2018.00053. URL: http://dx.doi.org/10.3389/fncir.2018.00053.

[7] W. Gerstner et al. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014. Chap. 1.3 Integrate-And-Fire Models, 19.1 Hebb rule and experiments. DOI: 10.1017/CBO9781107447615. URL: https://neuronaldynamics.epfl.ch/online/index.html.

[10] E. M. Izhikevich. Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling. *Cerebral Cortex* **17**.10 (Jan. 2007), 2443–2452. ISSN: 1047-3211. DOI: 10.1093/cercor/bhl152. eprint: https://academic.oup.com/cercor/article-pdf/17/10/2443/894946/bhl152.pdf. URL: https://doi.org/10.1093/cercor/bhl152.

[12] W. Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks* **10**.9 (1997), 1659–1671. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(97)00011-7. URL: http://www.sciencedirect.com/science/article/pii/S0893608097000117.

[15] M. S. Shim and P. Li. "Biologically inspired reinforcement learning for mobile robot collision avoidance". *2017 International Joint Conference on Neural Networks (IJCNN)*. Anchorage, AK, USA: IEEE, May 2017, pp. 3098–3105. DOI: 10.1109/IJCNN.2017.7966242.

[16] M. Wahde. *Autonomous Robots*. Course, Chalmers University of Technology, Gothenburg. 2016.

[17] M. Wahde. *Biologically Inspired Optimization Methods - An Introduction*. WIT Press, 2008. ISBN: 9781845641481.

[18] X. Wang et al. A behavior controller based on spiking neural networks for mobile robots. *Neurocomputing* **71**.4 (2008). Neural Networks: Algorithms and Applications 50 Years of Artificial Intelligence: a Neuronal Approach, 655–666. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2007.08.025. URL: http://www.sciencedirect.com/science/article/pii/S0925231207003025.

[19] X. Wang et al. Mobile robots modular navigation controller using spiking neural networks. *Neurocomputing* **134** (2014). Special issue on the 2011 Sino-foreign-interchange Workshop on Intelligence Science and Intelligent Data Engineering (IScIDE 2011) Learning Algorithms and Applications, 230–238. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2013.07.055. URL: http://www.sciencedirect.com/science/article/pii/S0925231214000976.

[20] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE* **87**.9 (Sept. 1999), 1423–1447. ISSN: 1558-2256. DOI: 10.1109/5.784219.

A Robot parameters

Parameter	Value
Radius, R	0.2 m
Mass, M	3.0 kg
Moment of inertia, $\overline{I}$	0.2 kg m^2
Wheel radius, r	0.1 m
Gear ratio, G	2
Coulomb friction constant, f_C	0.008
Viscous friction constant, f_v	0.02
Electrical constant, c_e	0.0333
Torque constant, c_e	0.0333
Armature resistance, r_A	0.62 Ω
Maximum applied voltage, $V_{\max}$	12 volt
Damping coefficient, α	15
Damping coefficient, β	0.78
Odometer sigma, σ_o	0.0002
Compass sigma, σ_c	0.05
Sensor opening angle, γ	0.5 rad
Number of sensor rays	3
Sensor parameter, c_1	0.03
Sensor parameter, c_2	0.1

Table A.1: Robot and sensor parameters

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
DIVISION OF VEHICLE ENGINEERING AND AUTONOMOUS SYSTEMS (VEAS)
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY