Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Bachelorarbeit Kognitionswissenschaft

# Inferring Interactive Behavior using Recurrent Forward Models

Marco Kinkel

31. Juli 2019

**Gutachter**

Prof. Dr. Martin Butz
Kognitive Modellierung
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

**Betreuer**

Dr. Sebastian Otte
Kognitive Modellierung
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

**Kinkel, Marco:**
*Inferring Interactive Behavior using Recurrent Forward Models*
Bachelorarbeit Kognitionswissenschaft
Eberhard Karls Universität Tübingen
Bearbeitungszeitraum: 01.04.2019 – 31.07.2019

# Abstract

Sensorimotor control is one of the most crucial competences of all living and moving beings. The ideas of embodied cognition suggest that cognitive abilities are grounded on sensorimotor dynamics. According to the active inference principle, a model of sensorimotor anticipation can be learned by reducing free energy, that is maximizing Bayesian model evidence. This thesis is based on an active inference model, developed using a long short term memory network to infer future system states in order to show goal-directed behavior. The thesis examines whether the anticipation of perception can generally be learned and how it can be used to optimize action in interaction with the environment and especially other agents. The experiments performed confirm that sensor data can be anticipated and used to avoid colissions static or moving agents. Sensor gradients can however not control an agent on their own. The resulting model forms a basis for further extensions to enable additional abilities of social cognition within the paradigms of embodied cognition.

# Kurzfassung

Die Kontrolle der Sensomotorik ist eine der wichtigsten Kompetenzen aller lebenden und sich bewegenden Wesen. Die Ideen des *Embodied Cognition* schlagen vor, dass kognitive Fähigkeiten auf sensomotorischen Dynamiken basieren. Nach dem Prinzip der aktiven Inferenz kann ein Modell für sensomotorische Antizipation gelernt werden, indem die *Free Energy* minimiert wird, d.h. die Bayesian Modellevidenz maximiert wird. Diese Thesis entwickelt ein einfaches Modell aktiver Inferenz mithilfe eines *Long Short Term Memory* Netzwerks um zukünftige Systemzustände zu inferieren und damit zielgerichtetes Verhalten auszuüben. Die Thesis erforscht ob die Antizipation von Perzeption gelernt werden kann und wie sie genutzt werden kann um in Handlung in Interaktion mit der Umwelt und speziell mit anderen Agenten zu optimieren. Die durchgeführten Experimente bestätigen dass Sensordaten antizipiert und genutzt werden können um statischen oder beweglichen Agenten auszuweichen. Sensorgradienten alleine reichen allerdings nicht aus, um einen Agenten zu steuern. Das entwickelte Modell bietet eine Grundlage für Erweiterungen, um - unter den Paradigmen des Embodiments - zusätzliche Fähigkeiten der sozialen Kognition zu ermöglichen.

# Contents

## Chapter 1

# Introduction

In the late 20th century, cognition was thought to be computation over mental representations (J. Fodor, 1979) in a highly modular cognitive architecture, where cognition and perception are separable processes (J. A. Fodor, 1985). This view led to the development of Good old Fashioned Artificial Intelligence (GOFAI) which followed the idea that true intelligence can be achieved by processing discrete symbols and resulted in the development of unbeatable chess computers and powerful data analysis algorithms. These applications of GOFAI have some crucial commonality: they are no real-world scenarios. A chess game has a final and discrete set of rules and system states and - as well as the analyzed data - it is fully observable. In contrast, the problems agents face in the real world are highly continuous and the perceived information is noisy and versatile. For this reason, GOFAI is not able to solve even the basic real world problems like walking, recognizing objects or interacting with them. These problems on the quest to finding general intelligence led to a shift in paradigms, the embodiment turn: true bodily competence can accordingly not be achieved without considering body processes like sensorimotor interactions. The idea of embodied cognition is to start at the body and let competent behavior emerge. Biological or artificial agents are able to control their own behavior in dynamical and continuous environments without having a discrete symbol-like representation of their body and movements (see Pfeifer & Bongard, 2006).

The agent's body and the dynamics between internal states, actions and perceptions play a crucial role in the understanding of cognition and lead to the free energy principle. According to K. Friston (2016) "the defining dynamics of any system [...] is that they [...] will move [...] so as to maximize [...] Bayesian model evidence." This means that the brain constantly generates predictions and explanations for its sensory inputs. Thus for all of the agent's states the model evidence must be maximized, which means that the free energy is minimized. "Free energy measures the quality of a model in terms of the probability that it could have generated observed outcomes" (K. J. Friston, 2016, p.105). The process "in which the brain tries to infer the causes of its sensory input while sampling that input to minimize uncertainty about its inferences" (Engel, Friston & Kragic, 2016, p.9) is

called active inference. The inference process is however not restricted to perception because also "action is regarded as the fulfillment of descending proprioceptive predictions" (K. J. Friston, 2016, p.101). This means that the brain constantly creates predictions of the consequences of our actions. By reducing the prediction error and thereby adapting the motor commands, these predictions are fulfilled automatically. The motor commands are thus adapted to fulfill the predicted and thus preferred consequences of action.
The active inference principle can be broken down to agents with low-dimensional inputs and actuators in a simple environment. To teach an agent an internal model of its own sensorimotor dynamics, its task must simply be to predict the outcome of an action and adapt its predictions to reduce the free energy, that is reducing the error between its predictions and the actual outcomes. The same principle works with a perception model of the world.
After learning a sufficient internal model the agent is able to perform active inference with its motor commands, as described above: to reach a goal position, the error between the predicted and the actual outcomes of some motor commands is minimized by adapting the motor commands which will eventually fulfill the predictions. Parts of this have already been implemented in previous works. Otte, Schmitt, Friston und Butz (2017) created a recurrent neural network (RNN) which was able to learn and encode the internal model of an agent's movement dynamics. Additionally the model can be used to perform prospective inference of motor commands to plan and act goal-directed. Otte, Hofmaier und Butz (2018) created a many-joint robot arm which is able to avoid collisions using its proximity sensors in prospective motor inference. However these RNNs did not learn a sensor model which would enable them to predict future sensor inputs, so it remains an open question whether it is possible to train the forward model to additionally predict the sensory consequences of action. This question is investigated in this thesis.

To answer not only the 'easy' questions of sensorimotor dynamics, the embodiment paradigm also examines cognitive capabilities of higher order, like social cognition. Other theories of social cognition stress the idea of a detached observer who interprets the explicit symbolic communication. Action-oriented theories of embodiment claim that social cognition always includes oneself and is a rather interactive process than a detached one. Especially simulation-theories claim that observed action is simulated by own internal models to understand other's intentions and goals. These theories are supported widely by neuroscience and psychological studies, as Barsalou (2008) outlines. Dominey et al. (2016, p.340) hypothesize that "learning and mastery of action-effect contingencies may also be critical for predicting consequences of actions from others and, thus, to enable effective coupling of agents in social contexts" (see also Di Paolo & De Jaegher, 2012). This means that the same sensorimotor principles must be applied to enable social interactive behavior.
Implementing a system that simulates observed actions with the internal model of

own sensorimotor dynamics is not the scope of this thesis. However an agent is designed to perceive other agent's movements and to predict the future behavior of the foreign agent. This is not done using a symbolic representation of the agent, but by anticipation of the own future sensor perceptions.

In this thesis a simple active inference model is developed with neural networks to infer future system states in order to show goal directed behavior. It focuses the prediction of sensor inputs to improve behavioral planning. The thesis examines, whether the anticipation of sensor inputs can generally be learned and how it can be used to optimize behavioral planning in interaction with the environment and especially other agents without having discrete representations.

# Chapter 2

# Foundations

## 2.1 Markov Decision Process

A Markov Decision Process (MDP) is a formalized behavior of an agent in a world that fulfills the Markov property, which means that no memory of previous states of the world is needed to represent the current state with all its properties. The properties of the world are the set of possible discrete states $\mathcal{S}$, the set of possible actions $\mathcal{A}$ that the agent is able to perform, the state transition probabilities $\mathcal{P}(s_{t+1}|s_t, a_t)$, that is the probability to reach state $s_{t+1}$ of the world after executing action $a_t$, given the world is currently in state $s_t$, and finally the reward function $\mathcal{R}(s_t, a_t, s_{t+1})$ which specifies a scalar reward that the agent will encounter after performing a certain action (Butz & Kutter, 2016, p.112f.).
The $Q$-function calculates the expected future reward following a behavioral policy $\pi : \mathcal{S} \to \mathcal{A}$:

$$Q^{\pi}(s, a) = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... \mid s_t = s, a_t = a, \pi] \tag{2.1}$$

where $\gamma$ is a discounting factor, $a \in \mathcal{A}$ is an action, $s \in \mathcal{S}$ is a world state and $r$ is the reward.
An optimal behavioral policy $\pi^*$ which tells the agent the best action for each world state, is the policy that maximizes all future rewards:

$$\pi^* := \text{argmax}_a Q^*(s, a) \tag{2.2}$$

The optimal values $Q^*(s, a)$ can be calculated with the famous Bellman equation:

$$Q^*(s, a) = E[r_{t+1} + \gamma \cdot \text{argmax}_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a] \tag{2.3}$$

However the real world can not be formalized with discrete, fully observable states. Our "sensory abilities give us hints about the actual state of the world, but never the true state" (Butz & Kutter, 2016, p.122). A formalism in which the agent decides about its actions as in an MDP but only in a partially observable environment is called partially observable Markov decision process (POMDP) (Astrom, 1965). The agent now has to deal with beliefs about states and update these beliefs after receiving additional information. The resulting optimal control problem

can also be solved with active inference, if the reward or cost function of the optimal control problem is translated to the cost function of active inference, namely reducing the prediction error (K. J. Friston, Samothrakis & Montague, 2012).

## 2.2 Inverse Anticipatory Behavior

The general idea of active inference has already been explained in the introduction. As mentioned above, optimal action in POMDPs depends on beliefs about states of the world. Our beliefs $B(s)$ about states $s$ of the partially observable world must be used to find an optimal action $a_t^*$ at any time step $t$. This is done by minimizing an energy function $F$ of beliefs about states of the world $B(s)$ while acting under a particular action policy $\pi$:

$$\pi^* = \text{argmin} \sum_t F(B(s_t)|\pi) \tag{2.4}$$

K. J. Friston (2017) suggests that the energy function $F$ should be the free energy function. Accordingly the goodness of an action policy consists of two components: the extrinsic value and the epistemic value. The extrinsic value is gained by minimizing the surprise about preferable future outcomes. If the agent's actions are expected to maximize the probability of outcomes that the agent prefers, the value of its actions is maximal when the surprise about the preferable outcomes is minimized. The epistemic value comes by the degree of how the agent's uncertainty of the states of the world will be reduced by the action policy. The less mutual information exists between a cause (e.g. the action) and a consequence (the perception), the more uncertainty is reduced by this action, the higher is its epistemic value. So the goodness of a policy is a classical exploration-exploitation trade-off, but both values depend on the minimization of surprise between the internal model of the world and the associated perception. This is done actively by pursuing action policies that minimize free energy.

The idea that anticipated consequences guide the actions is called inverse anticipatory behavior. A preferred state - action state, internal state or perception state - is achieved by anticipating the consequences of motor activity and adapting this activity in order to reduce the error to the desired effect (Butz & Kutter, 2016, p.144). The principle of inverse anticipatory behavior is implemented in this thesis using artificial neural networks.

## 2.3 Artificial Neural Networks

Artificial Neural Networks (NN) have proven to be successful for many applications like recognition, classification and prediction. In general a NN consists of units

with thresholds and activation functions (representing the neurons), and weighted connections between them (representing synapses). Structurally ANNs can differ widely, from one-directional feed-forward networks (e.g. McCulloch & Pitts, 1943) via convolutional NN (e.g. LeCun et al., 1989) through to recurrent neural networks (RNN), which are used in this thesis and will thus be described later.
A simple feed-forward NN is structured as shown in figure 2.1. As illustrated in figure 2.2 the activation $a_j$ of one unit $j$ is calculated by a (typically nonlinear) activation function $f_{act}$ applied to the net input of the cell, which is the activations of all $i$ preceding cells multiplied by the connection weight:

$$\begin{aligned} net_j &= \sum_i w_{i,j} \cdot a_i \\ a_j &= f_{act}(net_j) \end{aligned} \tag{2.5}$$

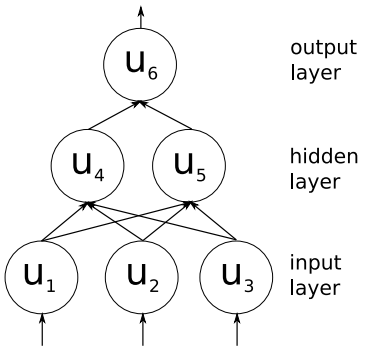


**Figure 2.1:** General structure of feed forward NN

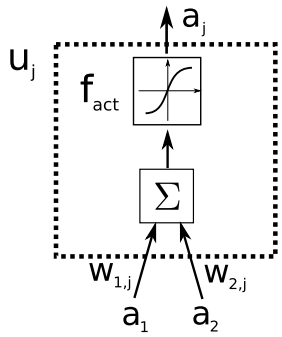


**Figure 2.2:** Cell activation in NN

## 2.4 Backpropagation

One reason for the success of ANNs is that they are able to learn complex nonlinear patterns. Backpropagation is one method to perform supervised learning, that is training the network with predefined patterns and applying learning rules until it outputs the wished targets (Bishop, 2006). In backpropagation the loss $\mathcal{L}$ is calculated as the difference of the network's output $o$ and the target $t$, typically as the mean squared distance:

$$\mathcal{L} = \text{MSE}(o, t) = \frac{1}{n} \sum_i^n (t_i - o_i)^2 \tag{2.6}$$

This loss is propagated back to the individual connection weights by calculating the partial derivative. The general equation to calculate the weight update $\Delta w_{ij}$

for the weight $w_{ij}$ of the connection between neurons $i$ and $j$ is the learning rate $\eta$ times the gradient $\delta_j$ times the output of the previous neuron $o_i$:

$$\Delta w_{ij} = \eta \cdot \delta_j \cdot o_i \tag{2.7}$$

If neuron $j$ belongs to the output layer, $\delta_j$ can be directly calculated as the difference between the output and the target times the derivation of the neuron's activation function $f$. Otherwise, the gradient $\delta_j$ is calculated using the gradients of all following neurons $u_k$:

$$\delta_j = f'_{act}(net_j) \cdot \begin{cases} (t_j - o_j) & \text{if } u_j \text{ is output neuron} \\ \sum_k \delta_k \cdot w_{jk} & \text{otherwise} \end{cases} \tag{2.8}$$

Optimization is typically done with gradient descent, that is the weights are adapted by following the negative gradient and thus finding a minimum in the unknown loss function. In stochastic gradient descent, the gradient is not calculated by the whole training data set, but approximated on-line with a sample of the data. There are different implementations of SGD to deal with challenges of gradient descent. One challenge is to find an appropriate learning rate. A too small learning rate might not be able to jump out of a local minimum in the loss function, while a too large learning rate leads to oscillating jumps over the minimum. There are methods to adjust the learning rate accordingly, e.g. the delta-bar-delta algorithm which increases the learning rate linearly or decreases it exponentially when certain conditions are met (Jacobs, 1988). A very simple implementation of learning rate adaptation is to reduce the learning rate when the loss did not reduce for a certain amount of epochs. This method is used later in this thesis.
One way to deal with flat loss plateaus, where learning takes a lot of time is the use of a momentum factor $\alpha$. Similar to the physical momentum, the weight updates of previous time steps are taken into account to enable acceleration when going in the same direction of the loss function:

$$\Delta w_{ij} = \eta \cdot \delta_j \cdot o_i + \alpha \cdot \Delta w_{ij}^{t-1} \tag{2.9}$$

Another optimization is to use individual learning rates for each parameter. The Root Mean Square Propagation algorithm for example adapts the learning rate for each weight based the previous gradients. The adaptive moment estimation algorithm (Adam) extends this idea further.

## 2.5 Optimization with Adam

Adaptive Moment Estimation (Adam) is a powerful implementation of stochastic gradient descent. Here not only the first moment of the gradients (the mean $m_t$)

but also their second moment (the uncentered variance $v_t$) is used to adapt the individual learning rates (Kingma & Ba, 2014):

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \delta_t \\ v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot \delta_t^2 \end{aligned} \tag{2.10}$$

The moment estimates are then corrected for a bias to zero:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \end{aligned} \tag{2.11}$$

The factors $\beta_1$ and $\beta_2$ are exponential decay rates for the moment estimates. $\delta$ is the gradient that should be optimized on.
Since Adam combines the advantages of individual learning rates and momentum, it is in most cases a good choice for optimization, which was shown by Ruder (2016).

## 2.6 Recurrent Neural Networks

To process series of inputs, e.g. depending on time, a recurrent neural network (RNN) can be used (Rumelhart, Hinton, Williams et al., 1988). It is an extension of the already described feed forward NN in the way that recurrent connections between units are possible. This enables to unfold a network over time and add a time-dimension to the input vector. In each time step, a new input can be induced to the network which itself uses recurrences to keep internal information in the loop. So for each time step $t$ an input vector $i^t$ with dimensionality of the input layer can exist. This is used to perform the forward pass to calculate the output $o^t$ for each time step. The computation of gradients can be done with backpropagation through time (BPTT)(Werbos, 1988, e.g.), which is basically backpropagation on an RNN unfolded over time. For each time step $t$ there must be a target $T^t$ with the dimensionality of the output layer, which is used to calculate the loss $\mathcal{L}^t$ and the gradients with respect to each time step. Since the output $o^t$ depends on the output $o^{t-1}$, the gradients of time step $t-1$ do not only depend on the loss $\mathcal{L}^{t-1}$ but also on the gradients of the time step $t$. This is visualized in figure 2.3.

This cascading gradient computation that always uses the gradients of subsequent time steps leads to a problem called the vanishing gradient problem. For an RNN where only the very last output is compared to a target (e.g. because the target for intermediate time steps is not known), the gradient will propagate back through time, always applying the chain rule of derivation. This repeated multiplication of small gradients eventually leads to their vanishment. For gradients with a value above 1.0 it leads to a so called gradient explosion, which will equally disable the network's ability to learn.

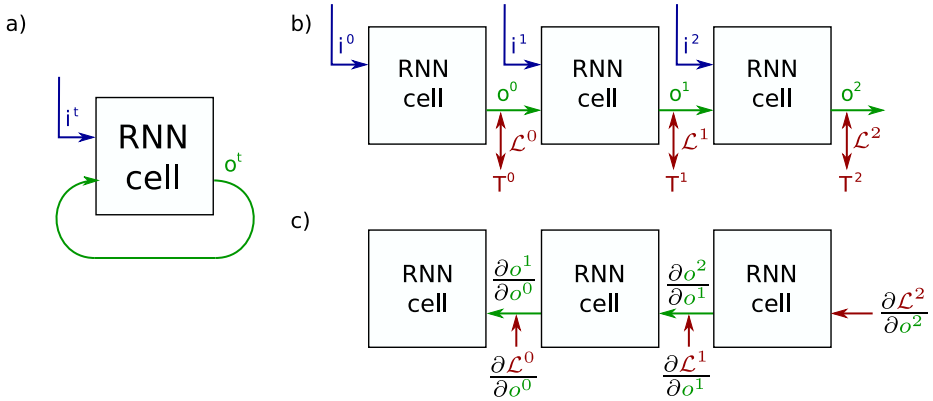


**Figure 2.3:** RNN visualized. **a)** The RNN in its recurrent form. **b)** RNN unfolded for three time steps with loss calculation. **c)** Backpropagation through time for the unfolded RNN.

## 2.7 Long Short Term Memory

The long short term memory (LSTM) by Hochreiter und Schmidhuber (1997) is an RNN with different gates of information flow. It is able to deal with the vanishing gradient problem because the gradients are allowed to flow unchanged to the preceding time step, so the gradients will not decrease as much.
The structure of an LSTM unit is visualized in figure 2.4 (compare Mallya (2017)).

It generally consists of the memory cell $c$ and three gates: the input gate $i$, the forget gate $f$ and the output gate $o$. Each gate has a trainable matrix $W$ that weights the inputs $x^t$ and a trainable matrix $U$ that weights the output of the previous time step $h^{t-1}$. Bias terms $b$ are added to each gate.

$$\begin{aligned} a^t &= \tanh(W_c x^t + U_c h^{t-1}) \\ i^t &= \sigma(W_i x^t + U_i h^{t-1} + b_i) \\ f^t &= \sigma(W_f x^t + U_f h^{t-1} + b_f) \\ o^t &= \sigma(W_o x^t + U_o h^{t-1} + b_o) \end{aligned} \tag{2.12}$$

The external input $x^t$ together with the recurrent output of the previous time step $h^{t-1}$ form the input vector for the unit. The input gate weights how much of the input is passed through to the cell. The cell updates its memory value accordingly, influenced by the forget gate which controls how much of the previous cell memory is kept.

$$c^t = i^t \odot a^t + f^t \odot c^{t-1} \tag{2.13}$$

The output gate finally controls the influence of the cell memory value to the unit output $h^t$.

$$h^t = o^t \odot \tanh(c^t) \tag{2.14}$$

In this thesis an LSTM will be used for the same reasons as in Otte et al. (2017): they are able to "associate temporally dispersed input events, which is essential when learning forward models in POMDPs. Moreover, LSTMs provide stable gradients over long time periods, which is critically important when complex control trajectories are required".

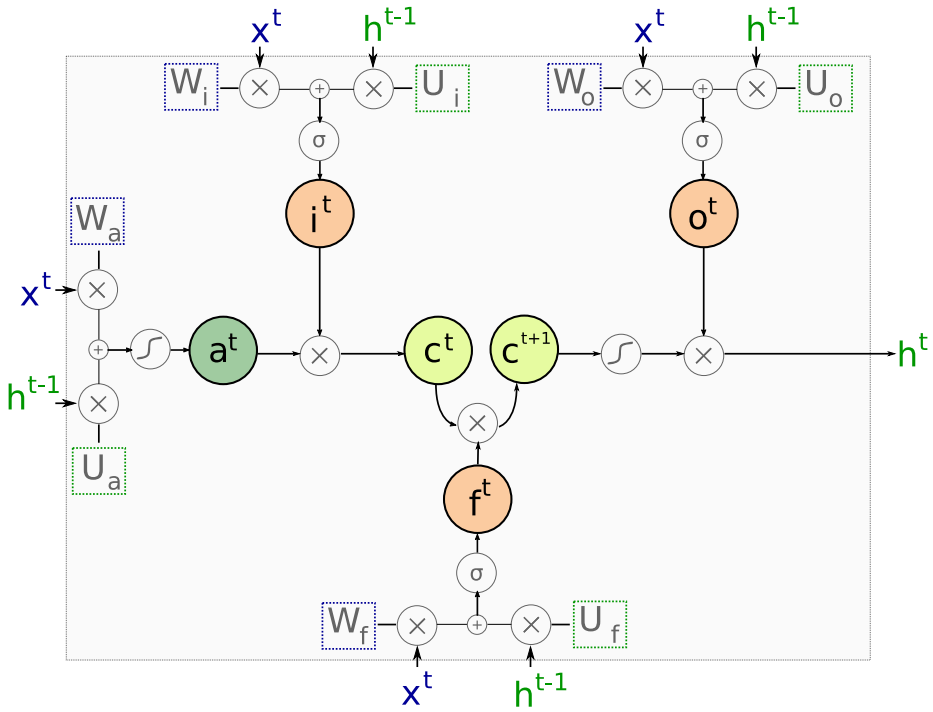


**Figure 2.4:** Structure of an LSTM unit.

# Chapter 3

# Approach

## 3.1 Environment

The general environment the model is trained in is a two-dimensional space, restricted by borders on each side. Like the RNN that simulates it, the environment uses discrete time steps. There is no gravity, but the agents are subject to inertia. The environment is physically simulated to compare the network's predictions to the simulated training data. The contact with borders and other objects leads to friction and a subsequent slowdown of the moving agent.

## 3.2 Agents

The agents that are controlled by the RNN are circle-shaped objects with four actuators (top-left, top-right, bottom-left, bottom-right). The actuators can have thrust values from 0 (no force) to 1 (maximal force) that push the agent in the opposite direction. The agents have positional sensors recording their change of position since the last time step. This information is hereafter called velocity $v = \Delta x$, since it is a change of position over a time period (although it is numerically not identical to the physically simulated velocity $v_{sim} = \frac{\Delta x}{\Delta t}$ with $\Delta t = 0.3$).

## 3.3 LSTM Network

The shape of the LSTM network depends in part on the stage of development that will be described in the next section. One LSTM cell unfolds over the task depending number of time steps with a hidden layer dimension of 32. Since the outputs of the network need to be in a specific dimensionality, an additional fully connected linear layer condenses the network's output to the desired size.

# Chapter 4

# Stages of Development

The thesis is structured in three stages of development. Each stage is designed to gain additional insight by extending the model for both the training and the motor inference procedure.

Stage 1 rebuilds existing models of prospective inference without sensor perception. The goal is to demonstrate that it is possible to use goal-directed behavior – planned by prospective motor inference – to coordinate multiple agents in interactive scenarios. Due to the lack of sensor information, the agents can not use own information on other agent's positions but rather must be steered by external goals. This way a chasing scenario can be constructed: agent $\mathcal{A}$ shall follow another agent B. Since $\mathcal{A}$ can not perceive $\mathcal{B}$ in any way, this behavior can only be exercised by setting $\mathcal{A}$ 's goal position to the current position of $\mathcal{B}$ in each time step. $\mathcal{A}$ is ought to perform active motor inference to reach this goal.

After showing that interactive behavior between agents is possible in the given environment, stage 2 introduces sensor information to the forward model. This new information is expected to improve the learning process. Whereas a collision during the learning process of stage 1 is completely unexpected, in stage 2 it can be attributed to the previously increasing sensor inputs. Unexpected changes of velocity can now be predicted by the sensory data. In the motor inference process, the additional sensor data is ought to be used in goal-directed behavior. If the objective is to reach a certain position, the agent should use its sensor inputs to avoid collisions and navigate around obstacles. If the quest is to seek proximity, the agent should try to maximize its sensor inputs by following other agents. However, in stage 2 agents are always provided with the real, physically simulated sensor data, instead of own sensor predictions. Although the certain knowledge of future sensor inputs is biologically not realistic, this approach aims to show the performance of the model with perfect sensor predictions. This can be used as baseline for the performance at stage 3.

The final stage 3 implements the anticipation of sensor information. Thus sensor gradients are now included in the training procedure, so the RNN-gradients are

calculated both with respect to the loss of position prediction and the loss of sensor prediction. During motor inference, the agent must now solely draw on its own predictions of future sensor information. These are ought to be used as described in stage 2, e.g. for obstacle avoidance or chasing behavior. The agent's performance compared now depends on its ability to anticipate sensor inputs.

## 4.1 Stage 1: Blind motor inference

### 4.1.1 Learning

In stage 1, the agent learns a model $\Phi$ of its own dynamics. The RNN encodes this model as approximation of the real partially unobservable dynamical system $\phi$.
The agent's current hidden state $\sigma^t$ is encoded in the internal state of the LSTM cell. Moreover the cell takes the velocity $v^t$, the acceleration $a^t$ and some predefined motor commands $m^t$ as inputs for time step $t$. By passing the inputs through the LSTM cell, its internal state is updated to $\sigma^{t+1}$ and it outputs the velocity $v^{t+1}$ and the acceleration $a^{t+1}$ after one time step:

$$(v^t, a^t, m^t, \sigma^t) \xrightarrow{\Phi} (v^{t+1}, a^{t+1}, \sigma^{t+1}) \tag{4.1}$$

The motor commands $m^t$ that serve as input for both the LSTM cell and the physical simulator in the learning procedure are required to resemble all possible movements of the agent. In the learning phase they must be defined externally, pursuing the goal to follow both stable movement paths with high velocities, as well as versatile and slow movement paths. With a chance of 70% the motor commands $m^t$ are designed to be equal to the commands of the previous time step $m^{t-1}$. Otherwise either no, one, two, three or all thrusts are active, each with a random activation value between 0 and 1. Each thrust is thus active with an average chance of 20%.
The outputs $v^{t+1}$ and $a^{t+1}$ are interpreted as the agent's prediction of the velocity and acceleration in the next time step. These values, together with the internal state $\sigma^{t+1}$ describe the predicted next state of the agent. According to Otte et al. (2017) the network makes more use of its recurrences when it does not predict the absolute position, but only a delta (that is the velocity). It is assumed that the dynamical system can be approximated better, the more derivations of the agent's movement the model learns, thus the acceleration is being added.
To approximate these predictions to the real velocity and acceleration, target data is needed. The weights of the RNN represent a model $\Phi$ that should approximate a real physical dynamical system $\phi$. This simulated environment provides the following target data: The physically simulated target velocity $v_{sim}$ is the change of the agent's real position $pos_{sim}$ after executing the given motor commands:

$$v^t_{sim} = pos^t_{sim} - pos^{t-1}_{sim} \tag{4.2}$$

In the same manner the acceleration is the change of velocity:

$$a^t_{sim} = v^t_{sim} - v^{t-1}_{sim} \tag{4.3}$$

This leads to a learning target of $(v^{t+1}_{sim}, a^{t+1}_{sim})$.
The loss $\mathcal{L}_{v,a}$ of one movement sequence consisting of $T$ time steps is calculated by

the mean squared error (MSE) of the predictions and the targets:

$$\mathcal{L}_{v,a} = \begin{pmatrix} w_v \\ w_a \end{pmatrix} \frac{1}{T} \sum_{t=0}^{T} \left( \begin{pmatrix} v^t \\ a^t \end{pmatrix} - \begin{pmatrix} v^t_{sim} \\ a^t_{sim} \end{pmatrix} \right)^2 \tag{4.4}$$

The weights $w_v$ and $w_a$ are used to compensate for numeric differences between the velocity loss and the acceleration loss. They can also be used to give the losses different priorities. It is for example more important to reduce the velocity error than to reduce the acceleration error. In this case the weights are set to $w_v = 100.0$ and $w_a = 0.01$ to compensate for numeric differences.
$\mathcal{L}_{v,a}$ is used to calculate the gradients for the weights and biases of the LSTM cell and the fully connected layer, which is done via backpropagation. An Adam optimizer with default parameters of $\beta_1 = 0.9$, $\beta_2 = 0.999$ and an initial learning rate of $lr = 0.01$ is used. During the learning procedure the learning rate is reduced by half if the loss has not reduced for the last ten optimization steps. This promises a more precise weight adaptation.
After optimizing the weights, the system continues to create random motor commands, predict their outcome and calculate the error.
Since the agents do not have sensors available, it is no use to create obstacles or foreign agents in the learning environment. It is not possible to predict a collision by any of the input data, so this sudden change of velocity and acceleration is not more than random noise.
One epoch of the learning procedure is described in algorithm 1 in the appendix section A and the general structure is visualized in figure 4.1.

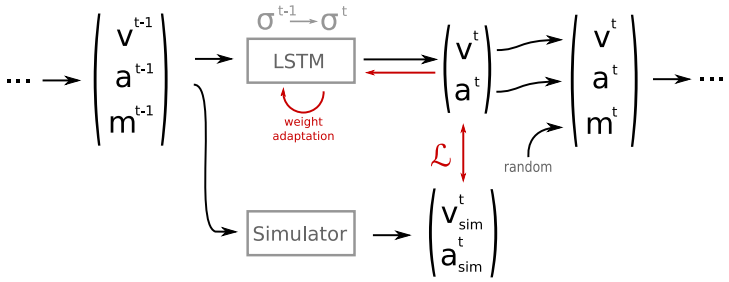


**Figure 4.1:** Model learning in stage 1

The learning procedure is performed for 200 epochs of respectively 450 time steps. The learning rate is initially 0.01 and decreases as described above. The results are shown in figure 4.2. One can see that the loss reaches a lower boundary after about 50 epochs already. Both the velocity and acceleration are learned sufficiently for

further experiments.

An observation during learning showed that the predictions were even in later epochs always wrong when the agent collided with a barrier. This is not surprising because no information is available that it could use to predict such collision. In stage 2 this observation should vanish due to the added sensor data.

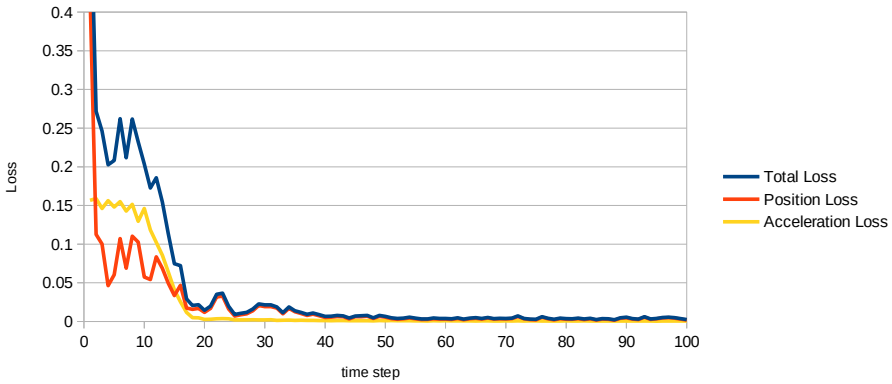


**Figure 4.2:** Learning result for stage 1

### 4.1.2 Motor inference

After a sufficient time of learning, the model is able to infer own motor commands in order to reach a given goal. To accomplish this, the agent initially executes random motor commands, predicts its new position, calculates the error to the given goal position and propagates it back to adapt the motor commands.
In each consecutive step the predictions of velocity and acceleration are used as input for the next time step. This way the network relies completely on its own internal model of the environment to adjust the motor commands in order to reach a goal position. After doing this for 15 future time steps repeatedly, the motor commands of the very next time step are executed in the environment. Now the real velocity and acceleration of this time step can be used again, because the agent is now in the next observable state. A pseudocode algorithm of one inference epoch is shown in algorithm 2 in the appendix section A.

The goal position $G$ is defined depending on the experimental scenario. The simplest way is to define it randomly within the boundaries of the environment.
The initial motor commands $m^0$ are set to be zero, so the agent is initially standing still in a predefined position $P^0$. After the first optimization step, the motor

commands will have changed and the agent moves towards the goal.
Since the network was trained to predict the velocity and the acceleration, these are also predicted in motor inference (see equation 4.11). However in motor inference only the velocity is used to calculate the position error $\mathcal{L}_G$, that is the squared difference between the predicted velocity $v^{t+1}$ and the target velocity $v^{t+1}_{goal}$ that would be needed to real the goal position $G$ from the position $P^{t+1}$. This position is the predicted position in the environment. Since the agent only predicts relative data, $P^{t+1}$ must be calculated by the current position $P^t$ and the predicted velocity $v^{t+1}$:

$$P^{t+1} = P^t + v^{t+1} \tag{4.5}$$

By proposition of Stoll (2019) the target velocity $v_{goal}$ should be clipped to prevent the agent from overshooting the position. Without velocity clipping, a large distance between $G$ and $P$ means that a high velocity $v_{goal}$ is needed to reach the goal position. The loss $\mathcal{L}_G$ and with that the motor command gradients will thus be too large to enable fine adjustments in motor coordination. Consequently the target velocity is clipped to a value of 0.015, which was shown to be most fitting in the experiments in chapter 5.
Overall we get the following calculation for the loss $\mathcal{L}_G$:

$$\begin{aligned} P^{t+1} &= P^t + v^{t+1} \\ v^{t+1}_{goal} &= G - P^{t+1} \\ v^{t+1}_{goal} &= \text{clip}(v^{t+1}_{goal}, 0.3) \\ \mathcal{L}_G &= \text{MSE}(v^{t+1}, v^{t+1}_{goal}) \end{aligned} \tag{4.6}$$

Note that the loss $\mathcal{L}_G$ does not describe a prediction error, because it is assumed that the model has sufficiently learned the dynamics of the system. Instead it describes the error to the goal position after applying the given motor commands. The motor inference process is visualized in figure 4.3

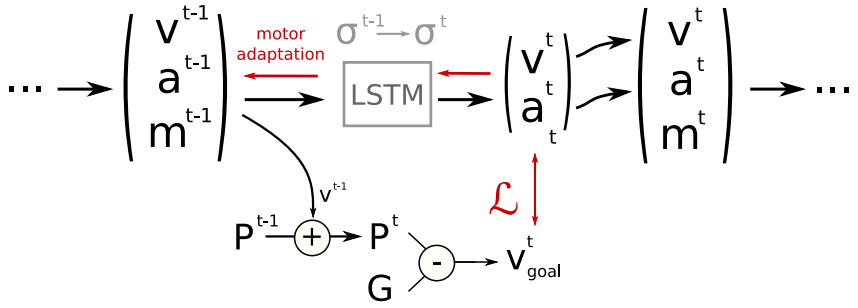


**Figure 4.3:** Motor inference in stage 1.

A first experiment is performed in order to examine the agent's capability of goal directed motor inference. Does the agent use its learned movement predictions to adjust the motor commands in order to reach a given goal? This is measured by the number of time steps the agent needs to reach and stay at a given goal position. The goal position $P$ is randomly picked within the borders of the environment. The initial agent position $P^0$ is set to the center of the environment. The target position changes after the agent reaches the target and stays within a distance of half its radius for ten consecutive time steps. After a target hat changed, the agent's position is not reset, instead the agent will continue at its previous position. Since the time needed to reach the goal depends on the distance to the goal, the measure is normalized by the initial distance between agent and target. The results are shown in figure 4.4.

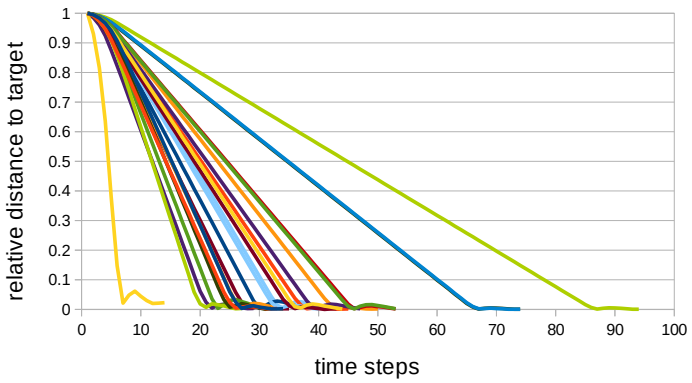


**Figure 4.4:** Distance to target in motor inference for stage 1

It is assumed that the initial distance between the agent and the target has an effect on the performance. The higher the distance, the more velocity can the agent get on its long way to the target. This likely leads to an overshoot and thus more time steps to reach a velocity where fine adjustments are possible again. However the data does not support this assumption, most probably because the velocity is clipped (see equation 4.6). To validate this assumption, the same experiment was performed without clipped velocities. The results (see figure 4.5) show a higher tendency to overshoot the target's position which confirms the assumption that velocity clipping is useful for fine coordination.

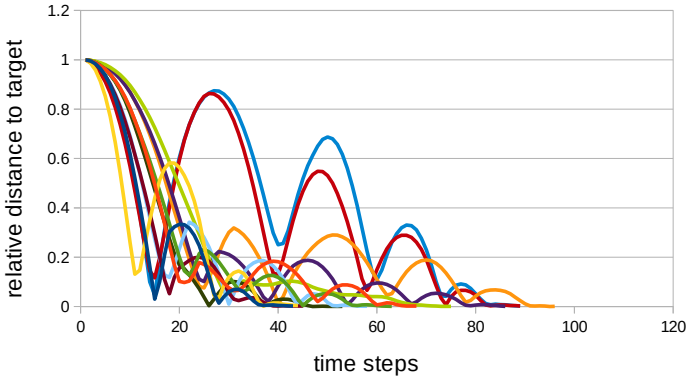


**Figure 4.5:** Distance to target in motor inference for stage 1 without velocity clipping

## 4.2 Stage 2: Simulated sensor data

Stage 2 is designed to examine how the agent can make use of sensor input. Sensors are added to the agents, measuring the sensor proximity $s$ as visualized in figure 4.6.

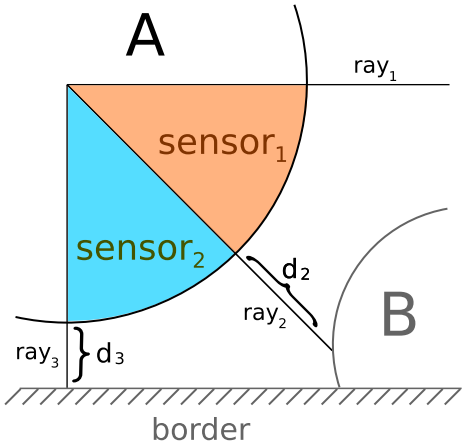


**Figure 4.6:** Sensor rays of agent $\mathcal{A}$ detecting proximitiy to agent $\mathcal{B}$ and the border

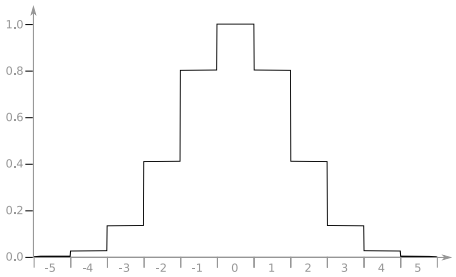


**Figure 4.7:** Gaussian point spread function for each sensor

Each of the 16 sensor rays is sent out in its particular direction. The directions are distributed circular around the agent depending on the number of sensors. Simple algorithms are used to calculate the intersection points with other agents (Weisstein, 2019a) and the border (Weisstein, 2019b). The sensor range of an agents with radius $r$ is set so that the distances $d_i$ of each sensor ray $i$ can take values between 0 (immediate closeness) and $d_{max} = 10 \cdot r$. To obtain a more practical scale, they are converted to a proximity value $p_i$ of ray $i$. If there is no intersection between the sensor ray and another agent or a border within a distance of $d_{max}$, the ray's proximity value is set to zero. Otherwise the proximity has a value between 0 and 1:

$$p_i = \begin{cases} 1 - \frac{d_i}{d_{max}} & \text{if } d_i \leq d_{max} \\ 0, & \text{otherwise} \end{cases} \tag{4.7}$$

The activity $s_i$ of the $i$th sensor is the higher proximity value of its surrounding rays:

$$s'_i = max(p_{2 \cdot i-1}, p_{2 \cdot i}) \tag{4.8}$$

To avoid surprising jumps in the sensor activity, a simple population coding is introduced by using a Gaussian-like point spread function to distribute the activity of each sensor to its neighbors (see figure 4.7):

$$\Psi(x) = exp\left\{-\frac{x^2}{2\sigma^2}\right\} \text{ with } \sigma = 1.5 \tag{4.9}$$

Overall the activity $s_i$ is the convolution of the sensor activity with this point spread function:

$$s_i = \sum_j s'_j \cdot \Psi(j) \tag{4.10}$$

### 4.2.1 Learning

While learning its dynamics, the agent now is allowed to use sensor information. This enables it to learn the effects of collisions to its velocity. The model of stage 1 is thus extended by the sensory input (see figure 4.8).

$$(v^t, a^t, m^t, s^t, \sigma^t) \xrightarrow{\Phi} (v^{t+1}, a^{t+1}, \sigma^{t+1}) \tag{4.11}$$

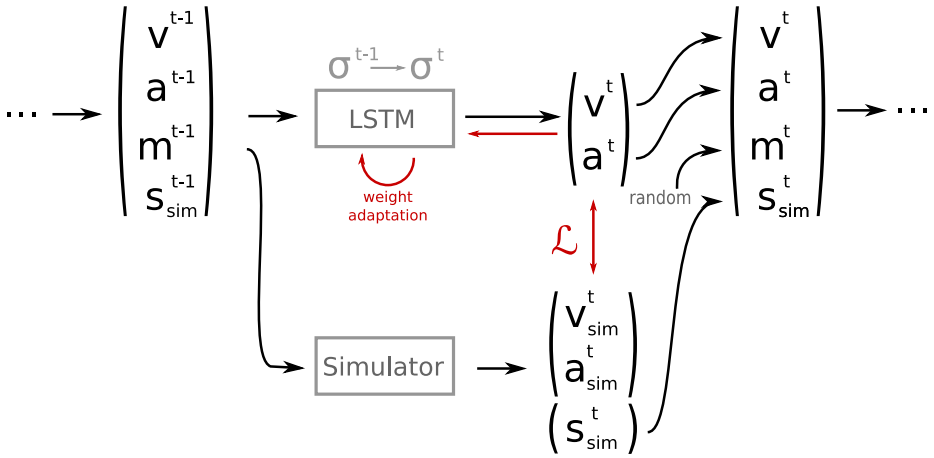


**Figure 4.8:** Learning procedure in stage 2

In order to learn the dynamics of collisions with borders and with static and moving obstacles, learning takes place in 200 epochs of respectively 30 mini epochs of 15 time steps each with different scenarios concerning the number and behavior of obstacles. All designed scenarios are visualized in figure 4.9. The goal of these different scenarios is to let the agent learn both linear and nonlinear sensory changes. For each mini epoch the scenario will change in the order visualized in figure 4.9. However scenario a) is exercised after every other scenario to ensure that the agent learns to predict its own dynamics with high confidence.

The net still predicts its future velocities after applying randomly created motor commands, just like in stage 1. The calculation of loss and gradients is also similar, because no additional predictions are made. The sensor inputs for each time step are calculated by a physical simulation. They represent the best possible sensor input in the sense of precision. Stage 3 will use self-predicted sensor inputs, which

are ought to be an approximation of the here used simulated sensor data.

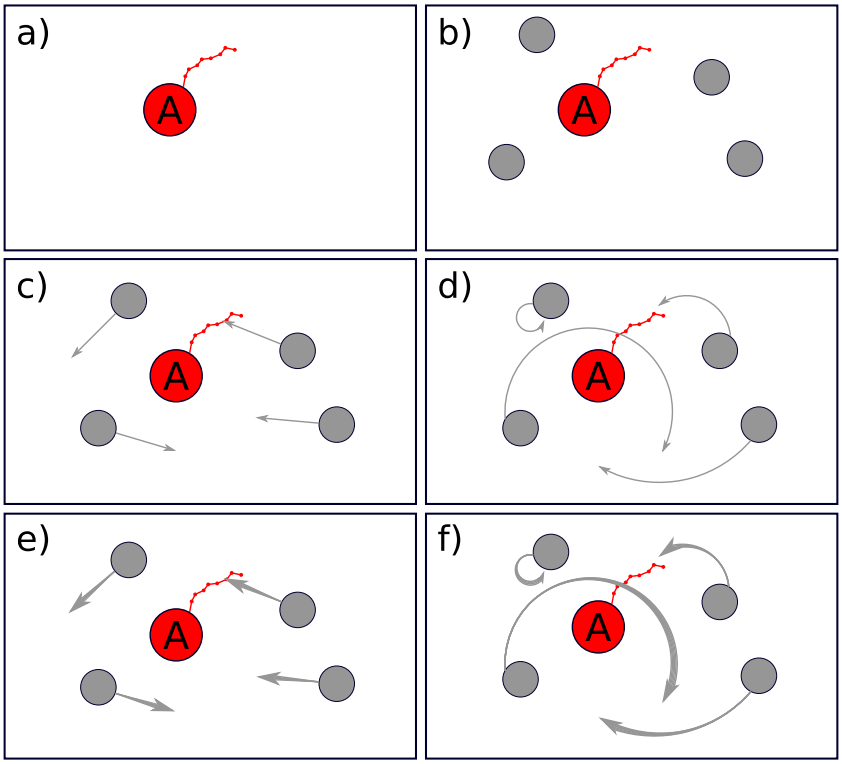


**Figure 4.9:** Scenarios for model learning in stage 2. **a)** Agent is alone. **b)** Static obstacles. **c)** Obstacles moving in straight directions with constant velocity. **d)** Obstacles moving on random sized circles with constant velocity. **e)** Like c) but with increasing velocity. **f)** Like d) but with increasing velocity.

The described learning procedure is executed for 200 epochs of 300 time steps. The resulting losses shown in figure 4.10 can not directly be compared to the losses in stage 1, because in stage 1 no obstacles were present. However one can note that the learning procedure now takes about 100 time steps to find a lower bound for the loss.

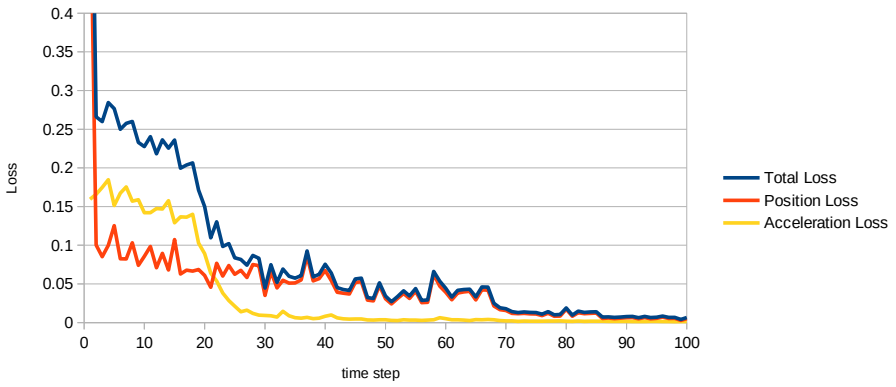


**Figure 4.10:** Learning results for stage 2

### 4.2.2 Motor inference

After successfully training the network, it should again be able to perform goal-directed motor inference. The agent still takes as input the current velocity $v^t$, acceleration $a^t$ and motor commands $m^t$, as well as its hidden state $\sigma^t$ and additionally the sensor data $s^t_{sim}$, which are calculated by a physical simulation as described above. The motor commands are randomly created as in stage 1. The inference process is visualized in figure 4.11.

After training the model with sensory inputs, it should perform better in prospective motor inference using this new source of information. However the better performance is assumed to be visible only in tasks where obstacles are included. Figure 4.12 shows the results for the non-obstacle inference task of stage 1 with the net trained in stage 2.

In comparison to the motor inference performance in stage 1 (see figure 4.4), the agent shows more overshooting behavior. This can be explained by the increased information entropy stored in the network's weights. Not only the effects of velocity, acceleration and motor data must be stored in the weights, but also the effects of sensor data. Plus, comparing the dimensionalities, sensor data make up for two

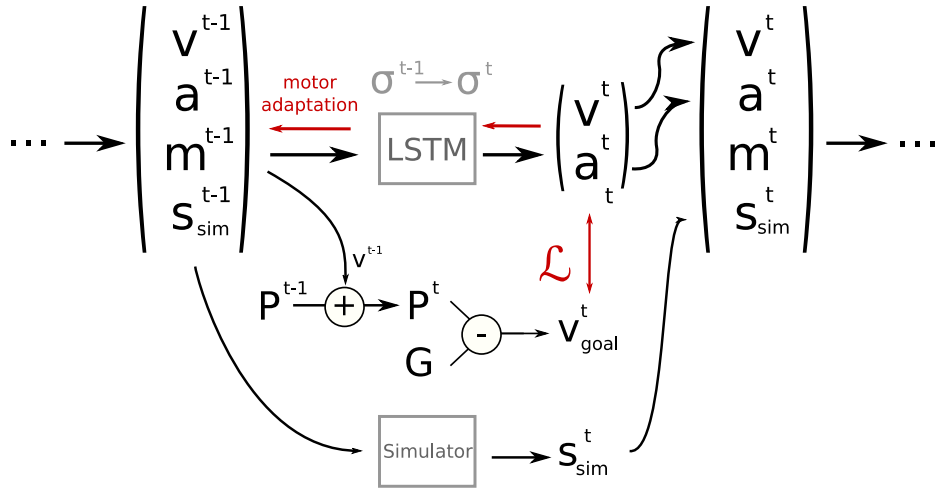


**Figure 4.11:** Motor inference in stage 2.

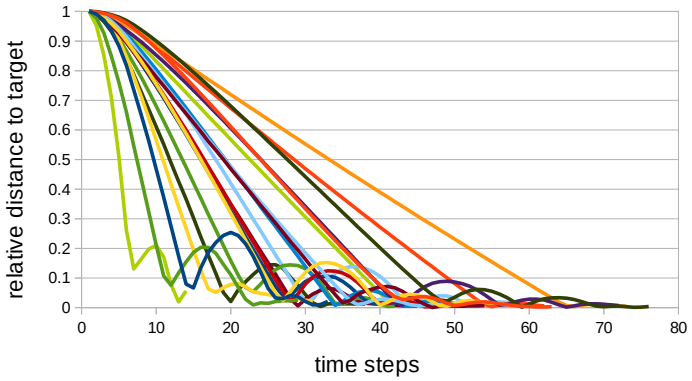


**Figure 4.12:** Distance to target in motor inference for stage 2

third of the input data. Therefore a slight imprecision in the input can have a large effect on the predictions.

To test the performance in presence of static or moving obstacles, the motor inference task is extended by obstacles with different random paths as in the learning evaluation (see figure 4.9). The results are shown in figure 4.13.

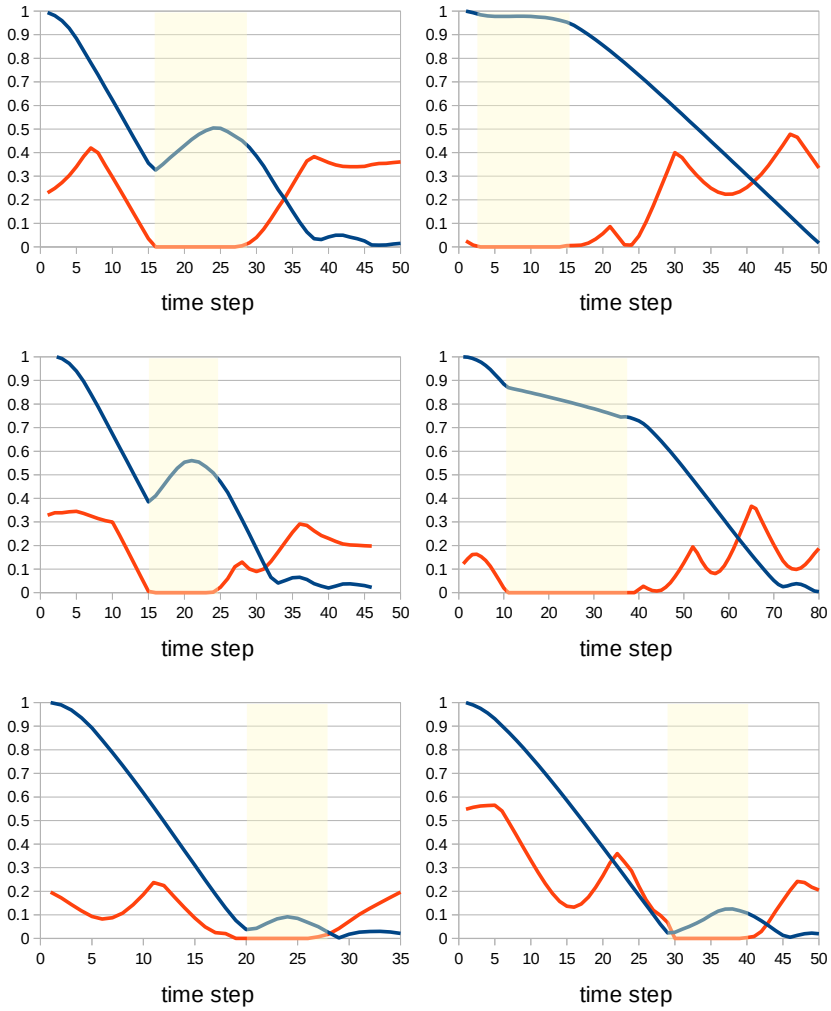


**Figure 4.13:** Performance of selected trajectories in motor inference for stage 2 with 20 obstacles on linear path. Blue line: distance to target (relative to initial distance). Red line: distance to nearest obstacle.

One can see clearly in the yellow marked areas that a collision often results in a disturbance of the agent's path to the target. By watching the trajectories it can be observed that the agent does not show any collision avoidance behavior. This is presumably because there exists only one target, for which the motor commands are inferred: reaching the target position. Adjusting the trajectory around an obstacle would mean not minimizing the distance to the target, but this is exactly what the network is designed to do.

As result sensory counter vectors (SCV) are introduced to enable avoidance be-

havior and bend the target velocities around an obstacle. SCVs are originally used in Otte et al. (2018) and Stoll (2019) for the same purpose. Since the target velocity $v_{sim}$ is the only thing upon the motor commands are optimized, the SCV must operate on this value. It moves the target velocity vector in the opposite direction of the sensor signal. For each of the 16 sensors an SCV

$$scv_i^t = -(s_t^i)^\beta \cdot o_i \tag{4.12}$$

is calculated where $o_i$ is the orientation vector of the sensor, $s_i^t$ is the sensor activity, i.e. the proximity after applying the point spread function. $\beta$ is an exponential weight factor for the sensory signal: the closer the obstacle gets, the higher must be the evading SCV. The overall SCV for one time step, independent of the sensor, is calculated as follows:

$$scv^t = \lambda \cdot scv^{t-1} + (1-\lambda) \cdot \gamma \cdot \sum_i scv_i^t \tag{4.13}$$

The first term of this equation is the by $\lambda$ weighted SCV of the previous time step to include the past sensory information. The sum of all individual SCVs is weighed by $\gamma$ to control the absolute numeric weight of the vector and thereby the influence on the agent's target. The overall SCV is added to the target velocity, before clipping the velocity vector and calculating the loss:

$$\begin{aligned} P^{t+1} &= P^t + v^{t+1} \\ v_{goal}^{t+1} &= G - P^{t+1} + scv^{t+1} \\ v_{goal}^{t+1} &= \text{clip}(v_{goal}^{t+1}, 0.3) \\ \mathcal{L}_G &= \text{MSE}(v^{t+1}, v_{goal}^{t+1}) \end{aligned} \tag{4.14}$$

This addition of the SCV can be visualized nicely as a shift of the target position in each time step, depending on the calculated sensor data in the respective step. Figure 4.14 shows a scenario of obstacle avoidance. Agent $\mathcal{A}$ is ought to reach the red target position, while $\mathcal{B}$ is a static obstacle. The red line shows $\mathcal{A}$ 's predicted velocity when executing the inferred motor commands. The numbered circles represent the target positions shifted by the SCV for each of the 15 predicted future time steps. One can observe that the target shift is bigger, the closer the agent moves to the obstacle, which is the preferred effect of the SCV.

Another useful addition to the network is the implementation of a movement orientation-based sensor sensitivity. This has already been proposed by Stoll (2019) as follows: the importance of a sensor's activity should depend on the orientation of the sensor compared to the movement direction of the agent. A sensor activity in the opposite movement direction is not as important as an activity in movement

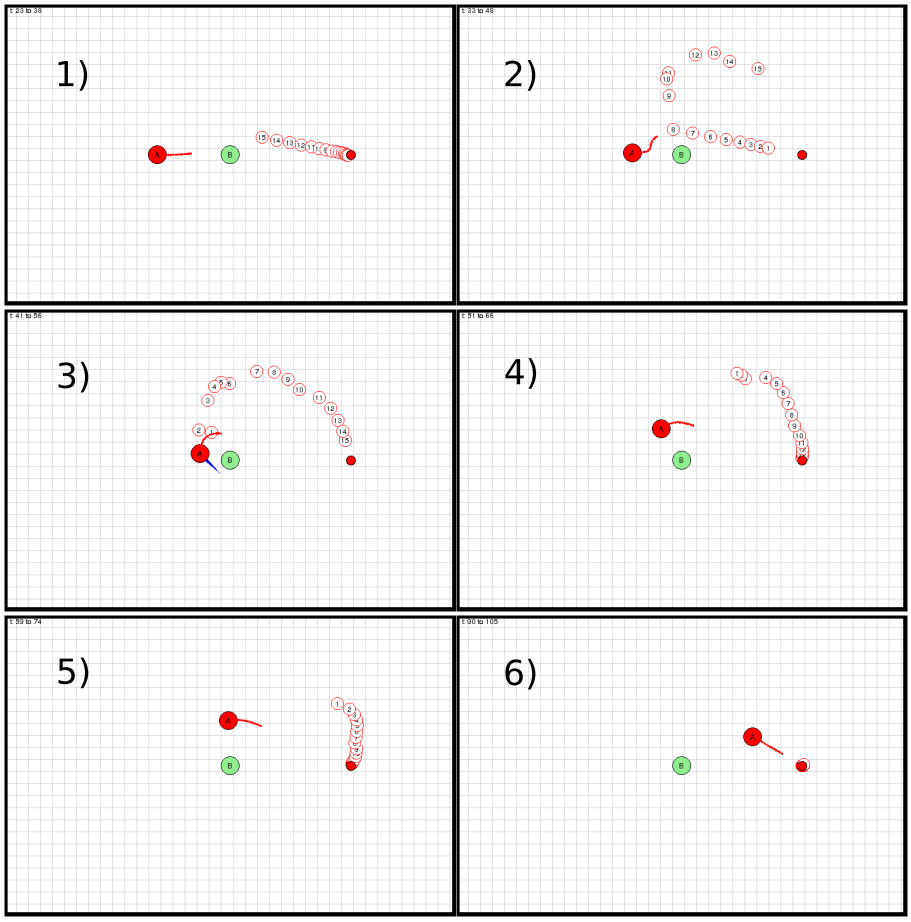


**Figure 4.14:** Selected time steps of obstacle avoidance behavior with SCV. The red line depicts the predicted future positions of $\mathcal{A}$ .

direction. Thus the sensor data is weighted with the cosine similarity of these orientations, shifted to the sensor range of $[0, 1]$. The sensitivity $sens_i$ of each of the $i$ sensors with the sensor orientations $o_i$ and the agent velocity $v^t$ of time step $t$ is thus calculated:

$$sens_i = \frac{1}{2}\left(\frac{v^t \cdot o_i}{\|v^t\| \cdot \|o_i\|} + 1\right) \tag{4.15}$$

This calculation is extended in this thesis by a component of target orientation. It sometimes occurs that the agent $\mathcal{A}$ collides with $\mathcal{B}$ and is carried in $\mathcal{B}$ 's direction (e.g. to the top). This shifts the orientation of A's velocity $v^t$ into the direction of $\mathcal{B}$ 's movement. Now to break loose of $\mathcal{B}$ , an SCV is needed that points away from $\mathcal{B}$ , back into the direction of the original target. This SCV is however not strong enough because the sensor signal is suppressed in the opposite orientation of $\mathcal{A}$ 's movement. Thus $\mathcal{A}$ 's perception of $\mathcal{B}$ is most weak and contributes too little

to the SCV (see figure 4.15). A solution to this is to take the orientation towards the target into account, calculate the sensor sensitivity for this orientation and use the mean of the velocity sensor sensitivity and the target sensor sensitivity (see figure 4.16). This addition reduced the number of time steps with collisions by a third in the scenarios where the original problem occurred.

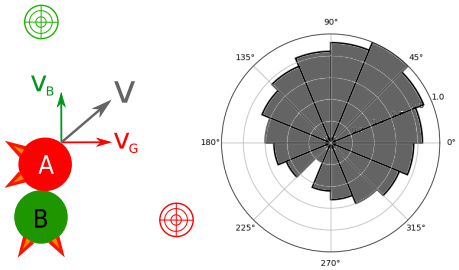


**Figure 4.15:** Velocity $v$ of $\mathcal{A}$ is calculated by the velocity $v_{\mathcal{B}}$ from being pushed by $\mathcal{B}$ and the velocity $v_G$ towards the goal. Using this velocity vector the sensors will have low sensitivity in the direction of $\mathcal{B}$ , which prevents strong SCVs.

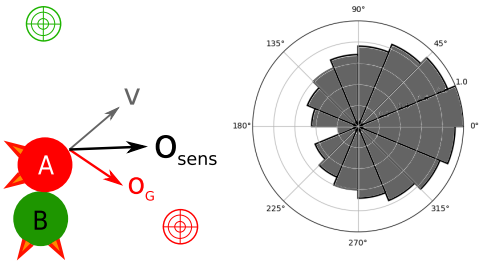


**Figure 4.16:** Extension of sensor sensitivity. Orientation $o_{sens}$ that is used to calculate the extended sensor sensitivity is calculated as the mean of the velocity orientation $v$ and the orientation towards the goal $o_G$. Using this new orientation vector the sensors will have higher sensitivity in the direction of $\mathcal{B}$ , which enables strong SCVs.

Observations of the agent's behavior without applying sensor sensitivity showed that the agent's path is influenced by the SCV even after the obstacle is already passed. On the contrary with sensor sensitivity, the agent returns to a straight goal-directed movement very quickly after an obstacle is passed. Figure 4.17 shows the same task as figure 4.14 but with activated sensor sensitivity.

A more isolated evaluation of the network's performance can be done in comparable experiments with designed tasks, instead of randomly defined positions. This is done in the following additional experiments.

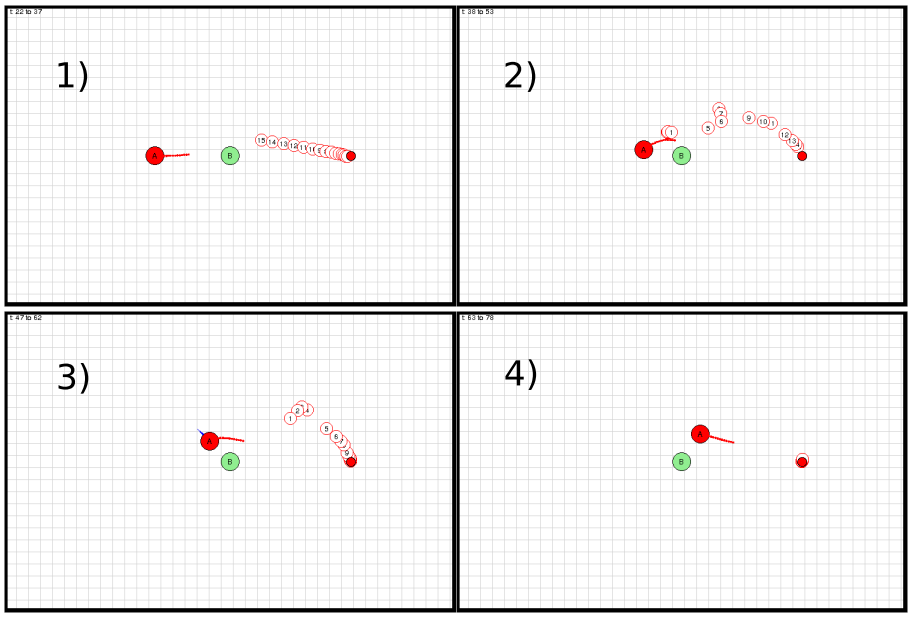


**Figure 4.17:** Selected time steps of obstacle avoidance behavior with SCV and sensor sensitivity. The red line depicts the predicted future positions of $\mathcal{A}$ .

## 4.3 Stage 3: Sensor prediction

This final stage implements the most innovative part of the thesis: the prediction of sensor data and its use for prospective motor inference. The model is now trained to predict sensor data in the learning phase. After sufficient learning this sensor information is ought to be used in the motor inference process. This prediction of sensor data is new and it is expected to perform approximately like the motor inference with simulated sensor data.

### 4.3.1 Learning

The model is extended by not only taking sensor information as input for the LSTM cell, but also by having them as output, interpreted as the predicted sensor input for the next time step:

$$(v^t, a^t, m^t, s^t, \sigma^t) \xrightarrow{\Phi} (v^{t+1}, a^{t+1}, s^{t+1}, \sigma^{t+1}) \tag{4.16}$$

This means, that in the learning phase the model needs additional targets to calculate the loss of the predicted sensor data $s^{t+1}$. This target is again calculated by the physical simulation, as shown in figure 4.18.

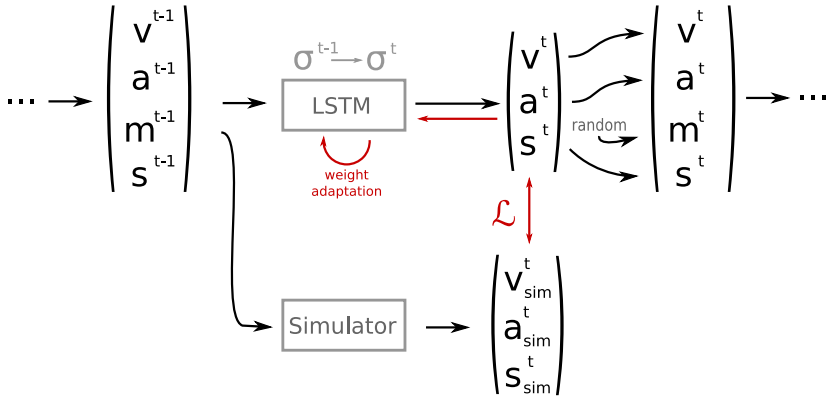


**Figure 4.18:** Learning procedure in stage 3

The learning procedure again uses different scenarios of obstacle movement to cover the full range of possible obstacle movements, including static and moving obstacles (see figure 4.9).

The net again applies random motor commands, predicts the resulting velocity, acceleration and now also the sensor input. From these predictions and the physically simulated target values, a loss is calculated, by which the network's weights

are optimized. Due to numeric differences, the loss values are weighted individually by $w_v = 100.0$, $w_a = 0.01$ and $w_s = 1.0$.

$$\mathcal{L}_{v,a} = \begin{pmatrix} w_v \\ w_a \\ w_s \end{pmatrix} \frac{1}{T} \sum_{t=0}^{T} \left( \begin{pmatrix} v^t \\ a^t \\ s^t \end{pmatrix} - \begin{pmatrix} v^t_{sim} \\ a^t_{sim} \\ s^t_{sim} \end{pmatrix} \right)^2 \tag{4.17}$$

Learning is again performed in 200 epochs of respectively 30 mini epochs for the different scenarios. Each mini epoch consists of 15 time steps. The learning rate is initially 0.01 and decreases as in the previous stages. The results are shown in figure 4.19.

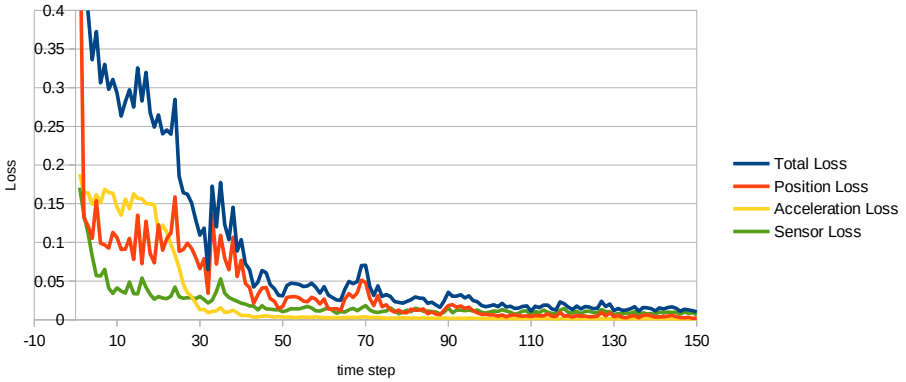


**Figure 4.19:** Learn result for stage 3

During learning it occurs that the network has a problem predicting the stimulus onset of sensor data. Even when there is no stimulus close by, it randomly starts predicting sensor inputs. Thinking about a simplified environment with no obstacles but the borders, this prediction behavior makes sense: there is no absolute information whatsoever about the agent's position in the environment, so there is no clue suggesting when a border will be perceived for the first time. The agent can not know whether it started at a position close or far to the right border, so if it moves to the right the border will turn up seemingly randomly. This stimulus onset problem will be approached in the following motor inference section.

To evaluate the performance in predicting sensor signals a simple task was designed where agent $\mathcal{A}$ passes the static agent $\mathcal{B}$ . The sensor predictions for selected time steps are shown in figure 4.20 for $\mathcal{A}$ and figure 4.21 for $\mathcal{B}$ . One column represents a time step. The image on top shows the environment at this certain time step. The sensor plots below represent the predicted sensor input (red for $\mathcal{A}$ , green for $\mathcal{B}$ ) and the physically simulated sensor input (black) for five time steps

of anticipation. The first thing to note is that the general activity of predictions is always lower than the real activity, which is due to the fact that high sensor activities are rare in the learning procedure. The second thing to observe is that the sensor predictions have a similar shape as the simulated sensor data, namely the gaussian shape. Note that this shape was not manually calculated subsequently but predicted by the network. This shows that the uncertainty of a sensor activation was adapted in the learning phase. Thirdly, the predictions in future time steps are of decreasing activity. This might be due to the fact that the general sensory stimuli in the learning procedure were decreasing because the agent moved away from the obstacles or vice versa. The decreasing activity can also be interpreted as "uncertainty" of predictions. The further in the future, the less input is available from the previous recurrent time steps in the LSTM cell. The most remarkable thing is the prediction of moving sensor stimuli. Not only the activation but the spatiotemporal change of sensor activity should be predicted. For the static agent $\mathcal{B}$ this task was probably easier because there was no additional uncertainties due to own movements. In the sensor plots of $\mathcal{B}$ one can see nicely that the predicted sensor activity changes in the direction of the real sensor change. For example at time step $20 + 1$ (which means the actual time step is 20 and the agent anticipated 1 step into the future), the most active sensor is between angles $[247.5°, 270°]$. As the prediction goes on into further future time steps, the activity of sensor $[270°, 295.5°]$ increases relative to the other activities. In time step $20 + 4$ it even exceeds all other activities, which is strong evidence that the movement is correctly translated to sensor predictions by the model. Agent $\mathcal{A}$ was also able to predict the direction changes of sensor activity, however they are not as close to the real sensor activity as for agent $\mathcal{B}$ , possibly due to the fact that $\mathcal{A}$ had to take its own movements into account.

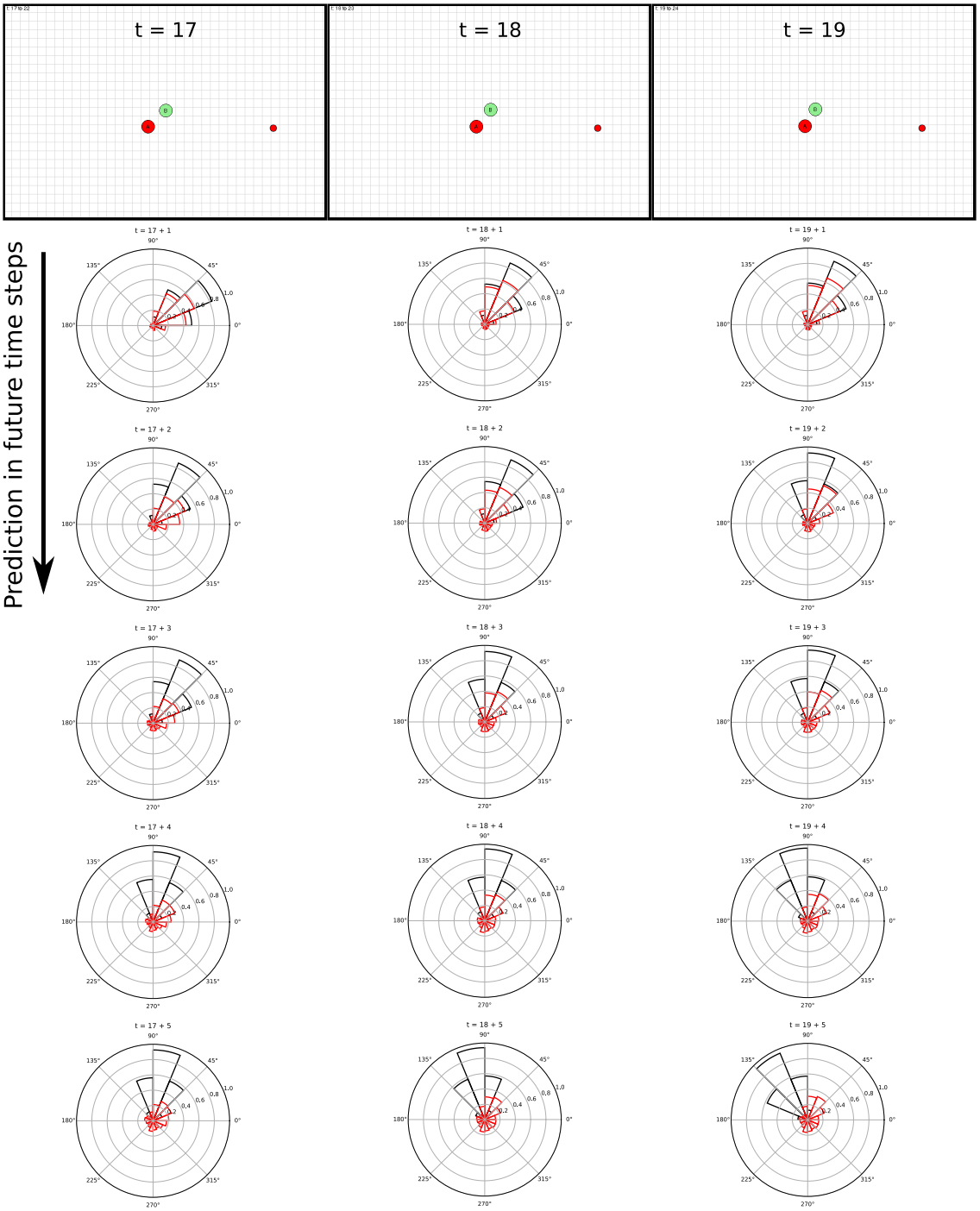


**Figure 4.20:** Sensor predictions of $\mathcal{A}$ in selected time steps. $\mathcal{B}$ is static while $\mathcal{A}$ moves to the red target. The sensor plots show the anticipated sensor inputs (red) and the simulated sensor inputs (black) of agent $\mathcal{A}$ . One column represents one time step.

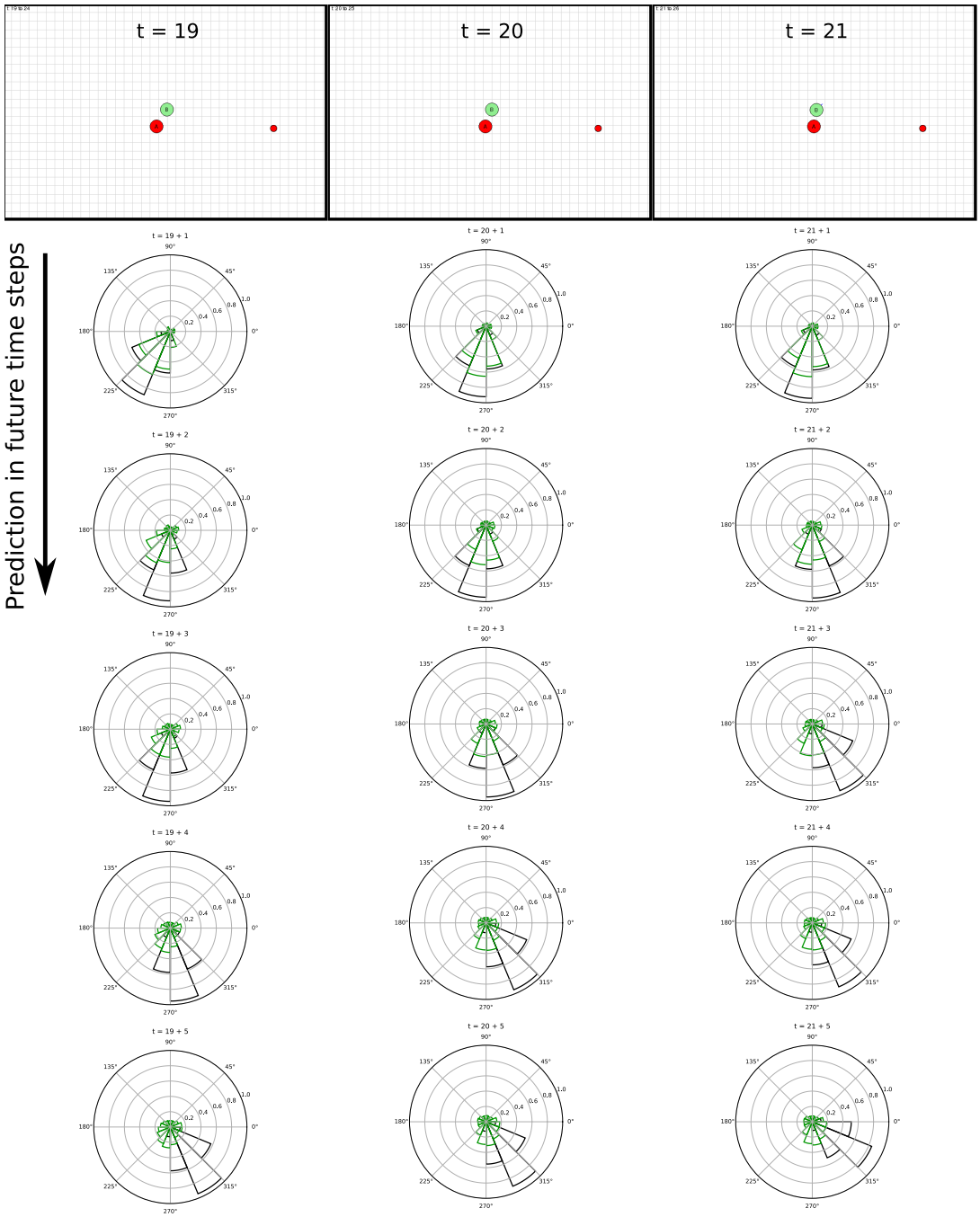


**Figure 4.21:** Sensor predictions of $\mathcal{B}$ in selected time steps. The sensor plots show the anticipated sensor inputs (green) and the simulated sensor inputs (black) of agent $\mathcal{B}$ . One column represents one time step.

### 4.3.2 Motor inference

The agent should now be able to perform goal-directed motor inference using its own sensor predictions. Now it relies not only on its predictions of velocity and acceleration, but also uses its own prediction of the sensor input it will encounter after performing certain motor commands. This and the predictions of velocity and acceleration are used as input for the network in the inference process. The motor commands for each time step are again initially zero, but adapted within the inference process.

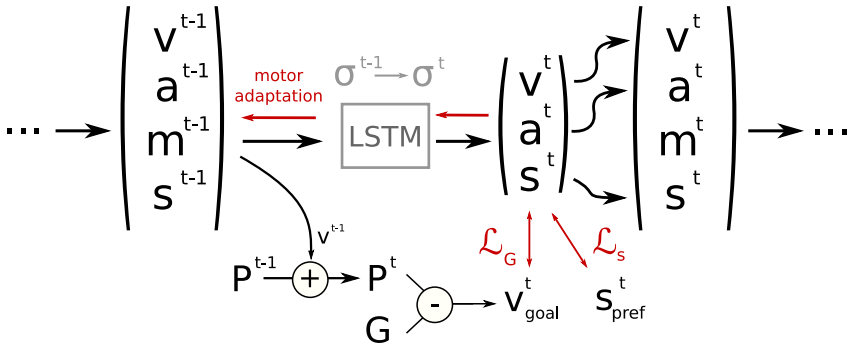


**Figure 4.22:** Motor inference in stage 3.

The loss can now be calculated with two components: $\mathcal{L}_G$ is still the difference between the current velocity $v^t$ and the velocity $v^t_{goal}$ needed to reach the goal position $G$. A new loss component $\mathcal{L}_s$ can be used if the network is ought to reach a preferred sensor input $s^t_{pref}$. This can for example be used to design an agent that prefers a high proximity to other agents or the border. $s^t_{pref}$ is then set to $(1, 1, ..., 1)$. On the other hand if the agents should avoid proximity, $s^t_{pref}$ is set to $(0, 0, ..., 0)$. The network will use the gradients of $\mathcal{L}_s$ and adapt the motor commands in order to reach this preferred sensor input. This makes it possible to replace the externally induced SCV by the more inherit sensor gradient: instead of manually changing the direction of the target velocity depending on the incoming sensory signal, the net can use motor inference and follow the gradient of the lowest sensor activity. This is done by setting a preferred sensor prediction of $s_{pref} = (0, 0, ..., 0)$ to calculate a sensor loss $\mathcal{L}_s = \text{MSE}(s^t, s_{pref})$, which is ought to be minimized by adapting the

motor commands. This approach is tested and compared to the SCV in section 5.2. By weighting the two loss components with $w_{\mathcal{L}_G}$ and $w_{\mathcal{L}_s}$ one can adjust the priority of the agent's task. If $\mathcal{L}_G$ is heavily weighted compared to $\mathcal{L}_s$, the agent will adapt its motor commands to reach the given goal position $G$ without caring much about the preferred sensor inputs $s_{pref}$. On the contrary, when $\mathcal{L}_G$ is weakly weighted compared to $\mathcal{L}_s$, the agent will concentrate more on its preferred sensor inputs $s_{pref}$ and adapt the motor commands accordingly.

To fix the stimulus onset problem, the agent's sensor predictions are manually set to zero, if the real sensor inputs in the previous time step were also zero. So if the agent did not encounter any real sensor perception in the last step, it is not allowed to predict any sensor input. This adjustment happens before $s^t$ is used for loss calculation:

$$s^t = \begin{cases} s^t & \text{if } s_{sim}^{t-1} \neq (0, 0, ..., 0) \\ 0, & \text{otherwise} \end{cases} \tag{4.18}$$

# Chapter 5

# Experiments

The following experiments are designed to show the difference in behavior and performance for different inference situations in presence of other agents or obstacles: 1) when the agent has no sensory information, 2) when the agent is provided with physically simulated sensory information and 3) when the agent uses its own predictions of future sensory information. The ability for interactive behavior, i.e. chasing and avoiding collisions is evaluated.

## 5.1 Experiment 1: Collision avoidance with motor inference

This experiment shows how well an agent avoids collisions when performing goal-directed behavior. Since the probability of a collision is fairly low in random set-ups, this experiment uses designed tasks. That is, the initial positions of both agents and the goal positions are preset to ensure that a collision course exists. The designed scenarios are visualized in figure 5.1. The measures are always taken for agent $\mathcal{A}$ , while agent $\mathcal{B}$ serves as obstacle. In each scenario, the initial position of $\mathcal{A}$ and the target are set from a circle around the obstacle $\mathcal{B}$ . All scenarios are tested from different starting positions of $\mathcal{A}$ , shown in figure 5.2. The colors of the starting positions are also used in the result plots below.

The two measures to examine the performance in collision avoidance behavior are the distance between $\mathcal{A}$ and its target and the distance between $\mathcal{A}$ and $\mathcal{B}$ . A distance of 0 indicates a collision.

The results for the collision tasks in stage 1 (plotted in figure 5.3) show that the agents $\mathcal{A}$ and $\mathcal{B}$ are on collision course in all designed tasks. Since there is no way for $\mathcal{A}$ to predict or avoid a collision, it steers directly towards the obstacle and collides with it, which can be observed at the straight distance lines which represent linear velocities. However $\mathcal{A}$ still manages to pass $\mathcal{B}$ after a certain amount of time because the motor commands and their gradients are not completely symmetric and thus the agent slides around the obstacle slowly. In the first scenario many

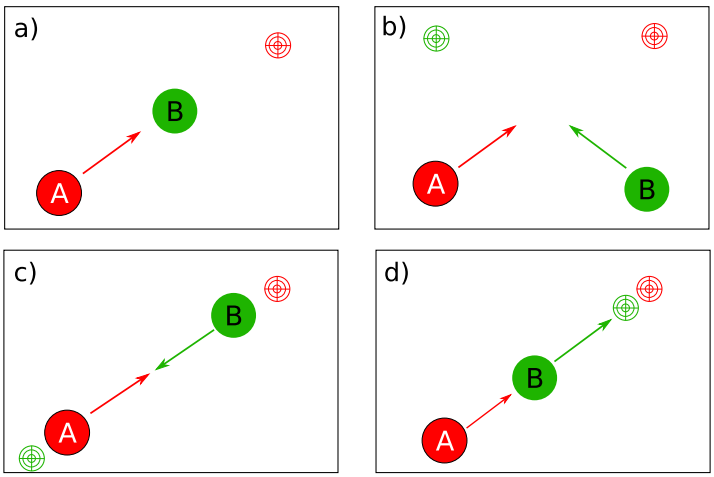


**Figure 5.1:** Scenarios to test collision avoidance behavior. Only agent $\mathcal{A}$ shows collision avoidance behavior. **a)** Position of $\mathcal{B}$ is fixed. **b)** $\mathcal{B}$ approaches a target on collision course.**c)** $\mathcal{B}$ slowly moves towards $\mathcal{A}$ 's initial position. **d)** $\mathcal{B}$ slowly moves in the same direction as $\mathcal{A}$

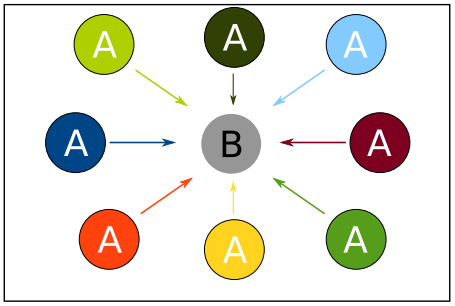


**Figure 5.2:** Colour explanation for results of collision avoidance tasks.

collisions occur between $\mathcal{A}$ and $\mathcal{B}$ . The challenge of stages 2 and 3 is to reduce the number of collisions and to reach the target quicker.

In stage 2 the agent uses additional sensor information to infer its future path. Now the SCV can be used to divert the target path around an obstacle. In this stage, the SCV is calculated by the physically simulated and therefore very precise sensor inputs. This allows for an early SCV onset, because the inputs can be simulated accurately far into the future: the agent knows exactly what sensory input it will encounter 15 time steps in the future, so the SCV can already adjust the goal velocity for this future time step, which again allows the agent to adapt its motor commands very early. The agents are therefore expected to collide very rarely.

The results for the four collision avoidance task shown in figure 5.4 clearly show

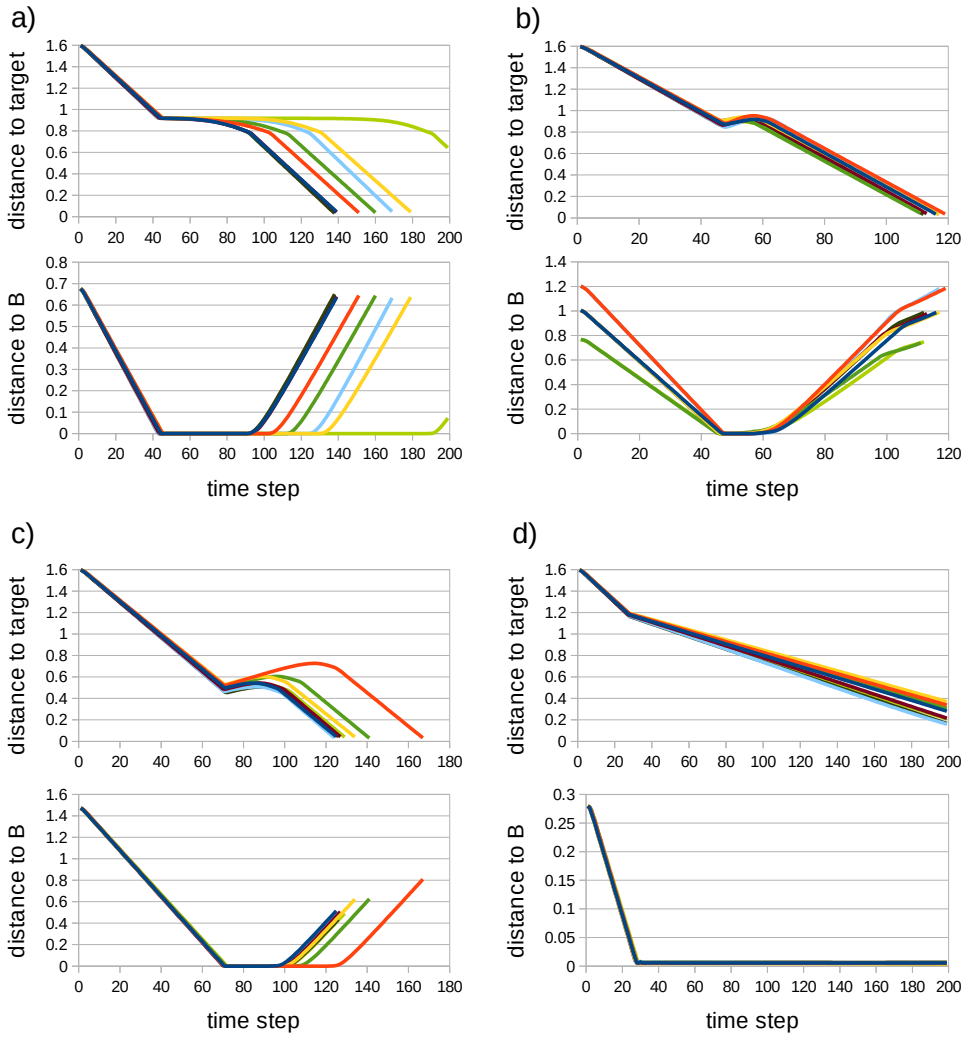


**Figure 5.3:** Results for motor inference collision tasks in stage 1.

that $\mathcal{A}$ shows avoidance behavior. In the scenarios $a$, $d$ and in the most cases of $b$ and $c$ the distance to agent $\mathcal{B}$ is never zero, which means that collisions are completely avoided. An example of how this successful avoidance behavior looks like is shown in figure 5.5.

However the results also show that in all scenarios there are cases in which the agent never reaches the target. This is due to an unfortunate constellation of the agents and the SCV. The SCV moves the target in the opposite direction of the sensor activity. If the target is in a line with the obstacle (which is the case in scenarios $c$ and $d$), the SCV will only slow down the instead of diverting its path around the obstacle. Only if an inaccuracy shifts the agent in one direction the SCV will lead it in the same direction. This is the reason why not all of the tasks

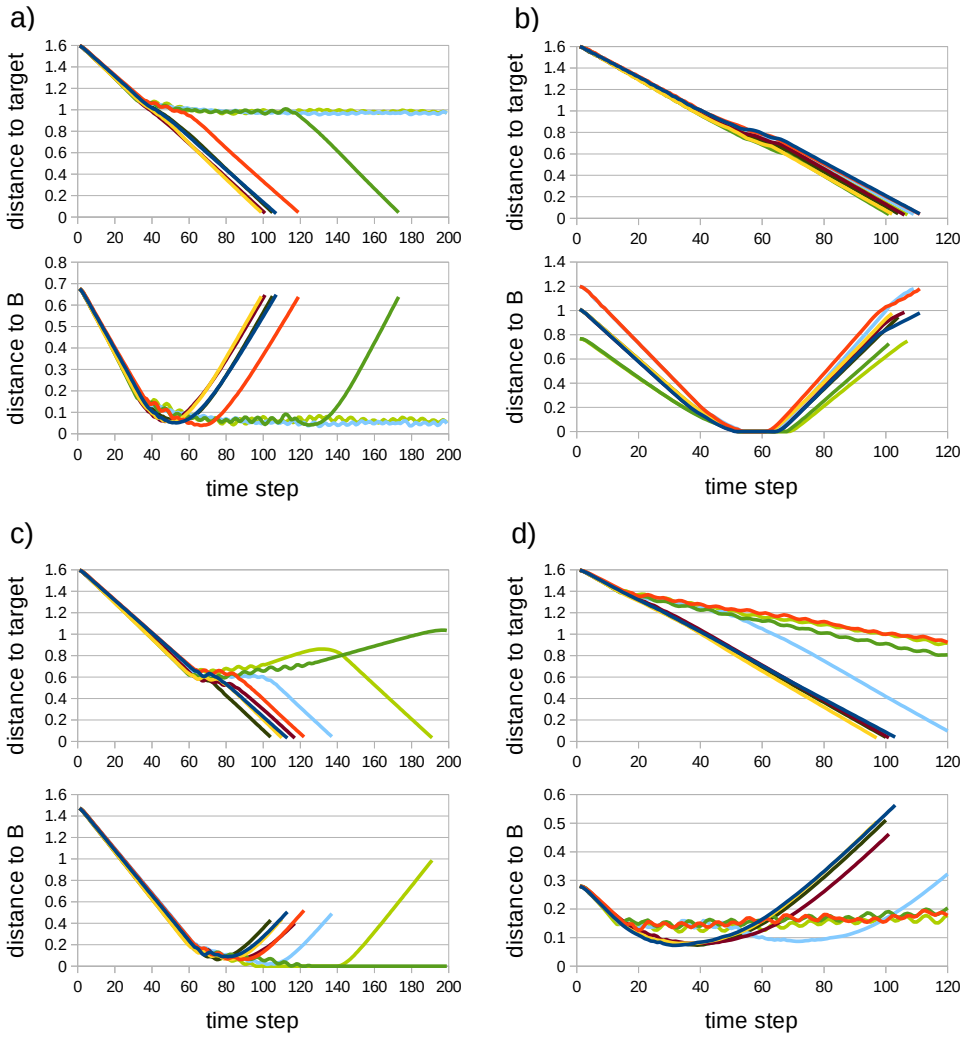


**Figure 5.4:** Results for motor inference collision tasks in stage 2.

in scenario *c* and *d* fail. An example of this failing SCV is shown in figure 5.6.

In stage 3 the sensor information that is used to calculate the SCV are not accurately physically simulated but predicted. Due to their contained inaccuracies, the avoidance performance is expected to be worse than in stage 2. On the other hand it is still expected to perform better than in stage 1 because the learned capability to predict sensor data is expected to suffice for a viable SCV. The sensor sensitivity introduced in section 4.2.2 is now used to weight both the sensor predictions and the measured sensor inputs.
The results for the collision avoidance tasks in stage 3 are visualized in figure 5.7.

In all tasks the agent is able to avoid collisions or at least to break loose from the obstacle B shortly after a collision occurred. Like this the agent is able to reach its

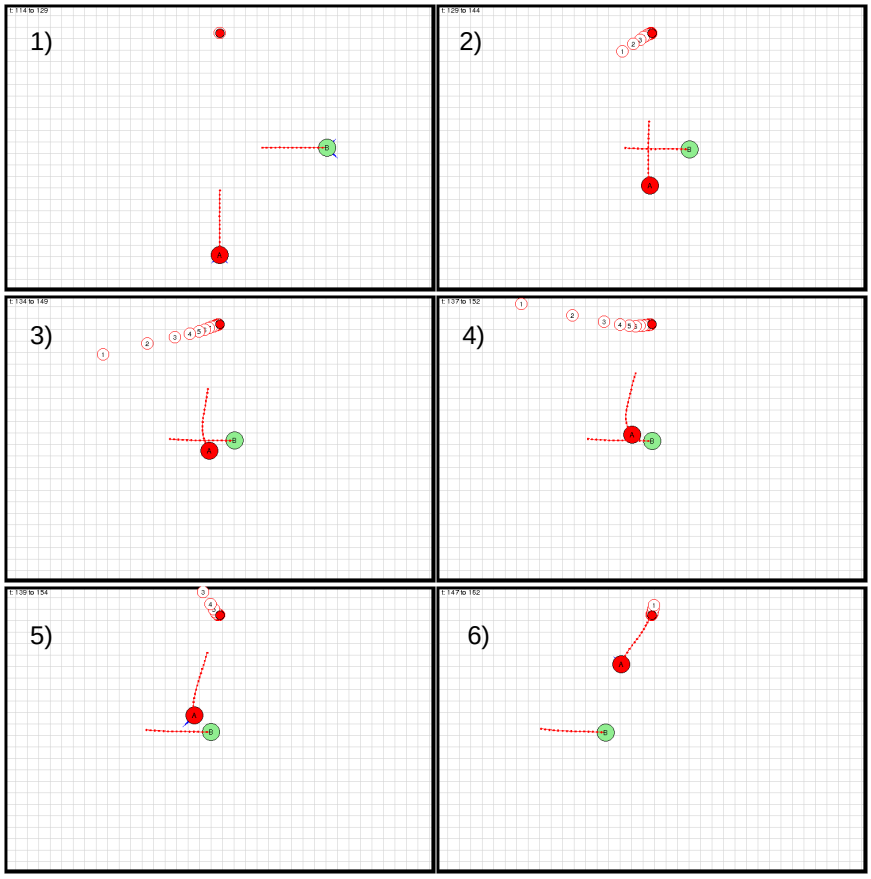


**Figure 5.5:** Successful obstacle avoidance behavior. The red lines depict the predicted movement path. The filled red dot is the target of $\mathcal{A}$ . The circled numbers represent the target adjustment by the SCV, with the numbers as future time steps.

goal quicker than in stage 1. It is also noticeable that there are again some tasks it which the agent does not reach the target. However this is only the case in tasks of scenario *d*, whereas in stage 2 this occurs also in the scenarios *a* and *c*. Against the expectations, stage 3 seems to perform better than stage 2 in this point. To compare the scenarios and stages, table 5.1 contains the summarized measures. The collision rate is the number of time steps in which a collision occurred. The other measure is the number of tasks (i.e. starting positions of $\mathcal{A}$ , see figure 5.1) in which the agent was able to reach the target within 200 time steps.
For scenario *a* the performance was very bad in stage 1, because $\mathcal{A}$ moved straight to the obstacle. In stage 2, all collisions were avoided with the use of the SCV. However, due to mentioned complications where the target and the obstacle are in a straight line, not all targets could be reached. This was however possible in stage 3, where the collision rate is between stage 1 and 2, as expected.
Scenario *b* is special because the agents move on collision course only for a short

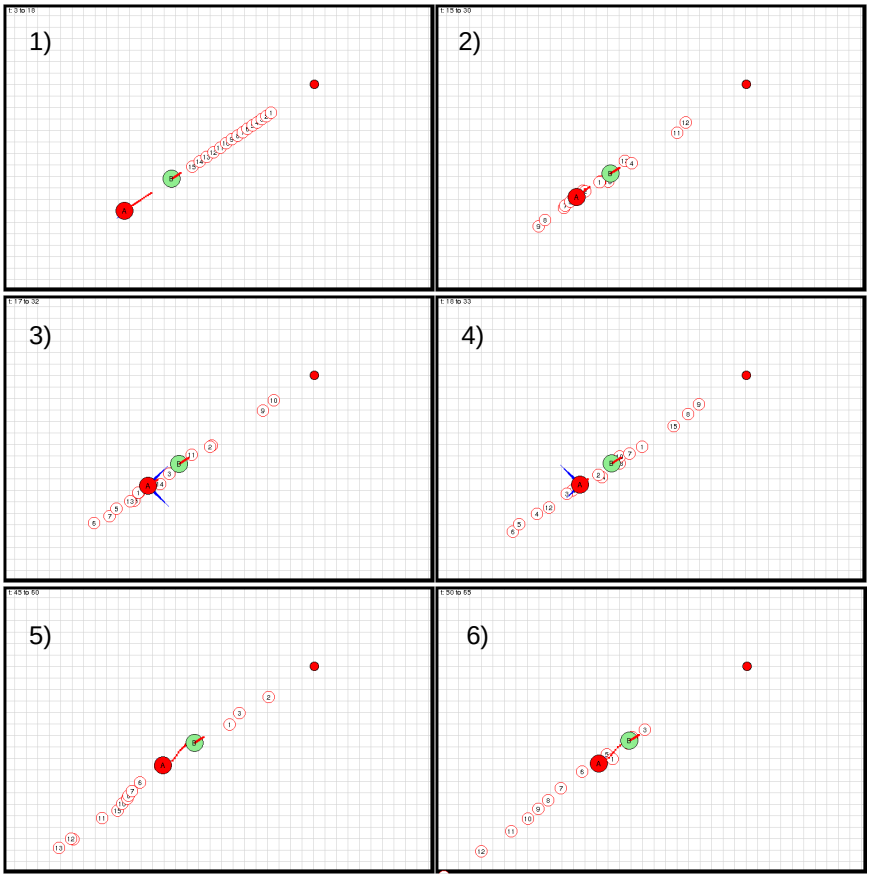


**Figure 5.6:** Failing obstacle avoidance behavior.

time. Due to $\mathcal{B}$ 's own movement in the orthogonal direction, it leaves $\mathcal{A}$ 's path without the need for collision avoidance, which is why the performance in stage 1 is not bad. Surprisingly it is even slightly better than the performance in stage 2. Observations show that in stage 2 $\mathcal{A}$ often tries to avoid a collision by moving in the same direction as $\mathcal{B}$ (an example is the path in figure 5.5). Stage 3 has the best performance in this task, probably due to a smaller SCV effect. While simulated sensor inputs (stage 2) have a relatively constant activity level, predicted sensor activities (stage 3) decrease with the passing of time due to prediction uncertainty. Since the SCV depends on the sensor activity, stage 3 thus has smaller SCVs, which reduces the avoidance movements and thereby the caused collisions in this scenario.
Scenario $c$ shows similar results as scenario $a$: stage 2 has the best results but not all targets were reached. Stage 3 follows directly, reaching all targets.
Scenario $d$ is very similar to this but the variance of performance varies strongly. While the target is never reached in stage 1, there are no collisions in stage 2. Again as expected, the performance in stage 3 lies in the middle between stage 1 and 2.

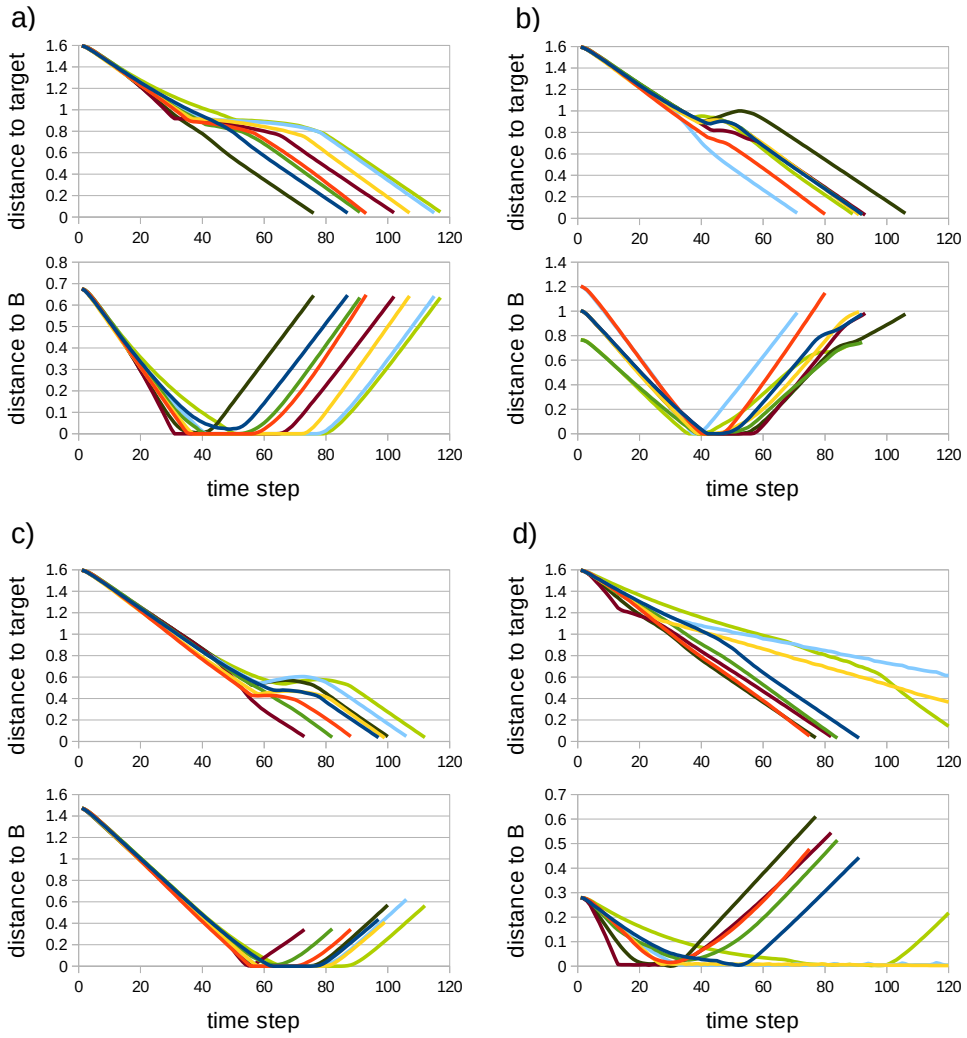


**Figure 5.7:** Results for motor inference collision tasks in stage 3.

In general the expectation that stage 3 performs better than stage 1 thanks to collision avoidance behavior but worse than stage 2 due to inaccuracies in sensor prediction, were fulfilled. Deviations could be explained by observing the agent's paths and identifying task-specific problems.

## 5.2 Experiment 2: Sensor gradient as SCV

This experiment examines whether it is possible to replace the SCV with the sensor gradient explained in section 4.3.2. Despite the numerical differences of factor 100 in the positional loss $\mathcal{L}_G$ and the sensor loss $\mathcal{L}_s$, they must be weighted with

| | Scenario a | | Scenario b | | Scenario c | | Scenario d | |
|---|---|---|---|---|---|---|---|---|
| | Collision rate | Target reached | Collision rate | Target reached | Collision rate | Target reached | Collision rate | Target reached |
| **Stage 1** | 47.9% | 7 of 8 | 11.4% | 8 of 8 | 25.6% | 8 of 8 | 86.4% | 0 / 8 |
| **Stage 2** | 0 % | 6 of 8 | 13.3% | 8 of 8 | 11.9% | 7 of 8 | 0 % | 5 / 8 |
| **Stage 3** | 11.9% | 8 of 8 | 7.9% | 8 of 8 | 12.8% | 8 of 8 | 33.1% | 7 / 8 |

**Table 5.1:** Results of collision scenarios for each stage. The red cells represent the worst performance for each scenario. The green cells represent the best performances.

| | Scenario a | | Scenario b | | Scenario c | | Scenario d | |
|---|---|---|---|---|---|---|---|---|
| | Collision rate | Target reached | Collision rate | Target reached | Collision rate | Target reached | Collision rate | Target reached |
| **SCV** | 11.9% | 8 of 8 | 7.9% | 8 of 8 | 12.8% | 8 of 8 | 33.1% | 7 / 8 |
| **Gradient** | 36.2% | 7 of 8 | 9.5% | 8 of 8 | 10.7% | 8 of 8 | 46.7% | 6 / 8 |

**Table 5.2:** Results of collision scenarios stage 3 with SCV and with sensor gradient as avoidance mechanics.

$w_{\mathcal{L}_G} = 1.0$ and $w_{\mathcal{L}_s} = 0.1$ in order to observe an effect of the sensor gradients to the movement path. $\mathcal{L}_s$ is thus by a factor of 10 higher than $\mathcal{L}_G$. An unwished effect of the sensor loss extension is that the velocity clipping loses its effect. As explained in equation 4.6 this caps only the target velocity used in the calculation of $\mathcal{L}_G$. The motor gradients calculated with $\mathcal{L}_s$ are not clipped, which leads to high and fast changing motor activity. Reducing $w_{\mathcal{L}_s}$ only adjusts the general effect of $\mathcal{L}_s$ to the movement path but does not cap the gradients. As shown in the results of this experiment, these comparably high motor commands do not diminish the performance. However one must be cautious when comparing the results to experiment 1 because the agent here has a bigger range of velocity.
The results are shown in figure 5.8 and compared with the performance of stage 3 with SCV in table 5.2. In scenarios *a*, *b* and *d* the obstacle avoidance using sensor gradients performs worse than using the SCV. However the agent still manages to find a path around the obstacles in most of the tasks. In scenario *c* the performance with sensor gradient is even better than with SCV. In general the results show that following the gradient of a preferred sensor stimulus is a usable technique for obstacle avoidance behavior.

## 5.3 Experiment 3: Chasing with motor inference

Two individually trained agents are involved in the chasing task. Agent $\mathcal{A}$ is in the same situation as in section 4.1.1: a random target is chosen every 50 time steps and $\mathcal{A}$ infers its motor commands to reach the target. Agent $\mathcal{B}$ is given the task to chase A. Again a benchmark measure is taken by performing the experiment

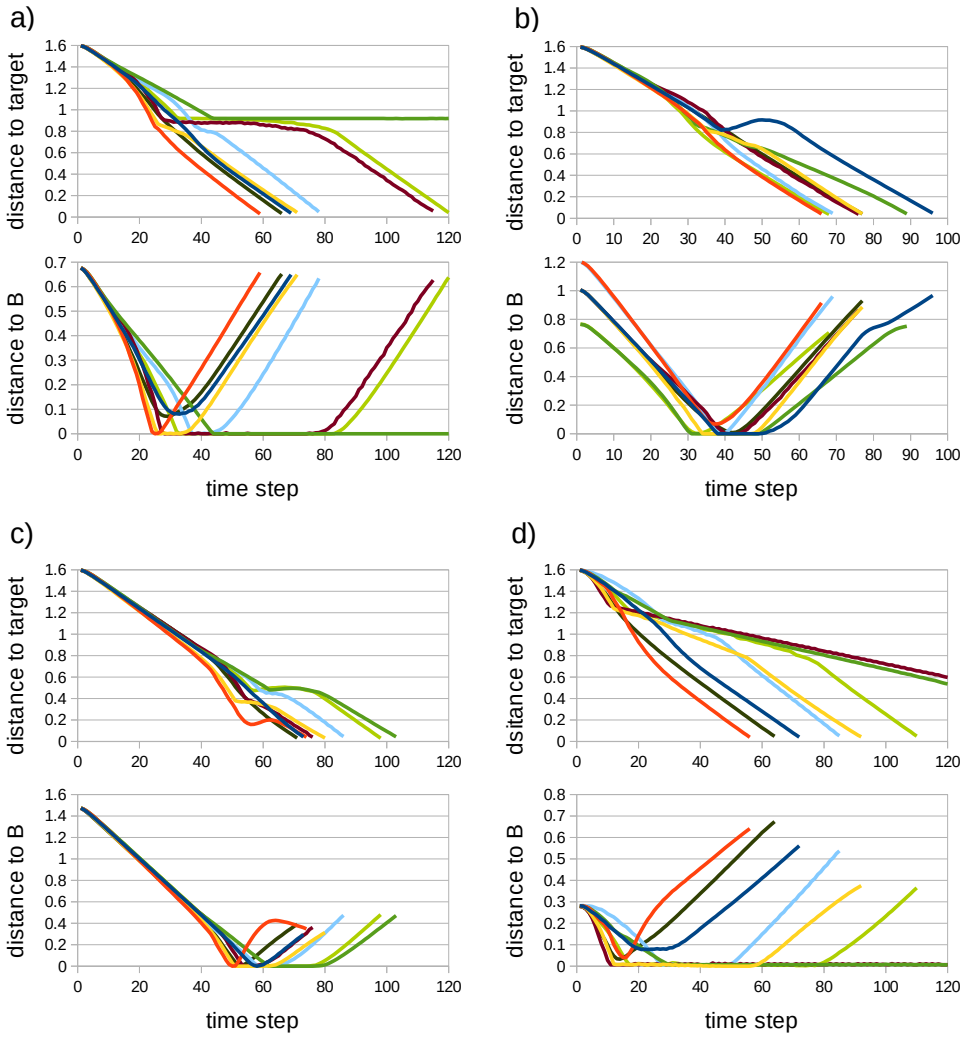


**Figure 5.8:** Results for motor inference collision tasks in stage 3 with sensor gradient avoidance.

in stage 1. Since there is no sensory data available in stage 1, $\mathcal{B}$ simply performs goal-directed motor inference like A. The goal position of $\mathcal{B}$ $G_B$ is always set to $\mathcal{A}$ 's current position, thus $\mathcal{B}$ follows a position gradient:

$$G_{\mathcal{B}}{}^{t} = P_{\mathcal{A}}{}^{t}$$

This means that $\mathcal{B}$ also takes the future positions of $\mathcal{A}$ as its future goal positions, which corresponds to a perfectly observable environment and therefore serves as best-case benchmark for the performance in stage 3.
The performance of $\mathcal{B}$ can be measured by the distance of $\mathcal{B}$ to $\mathcal{A}$ over time. A scalar measure to compare the performance is the average number of steps that

agent $\mathcal{B}$ is within a specific range of agent $\mathcal{A}$ during a certain amount of time steps. This distance is set to 0.3, which is five times the radius of an agent.
The experiment is done without other obstacles, because in stage 3 these would influence the sensor gradient and distract $\mathcal{B}$ from its original target. For the same reason the sensor perception of borders is disabled in this experiment.

The results for the position-based chasing task in stage 1 is plotted in figure 5.9.

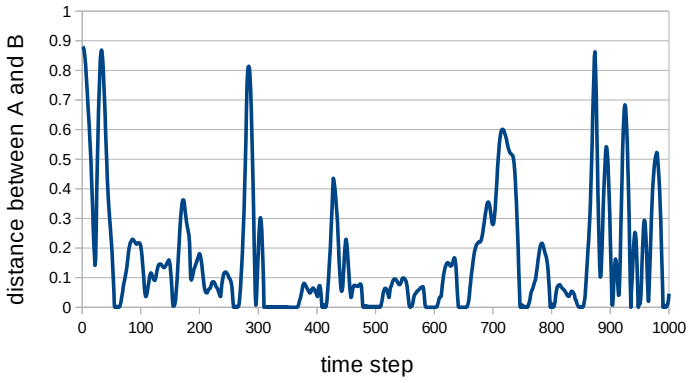


**Figure 5.9:** Distance between $\mathcal{A}$ and $\mathcal{B}$ in chasing behavior

In the recorded data, $\mathcal{B}$ is within the proximity range of 0 to 0.3 for 818 of 1000 time steps, so for about $\frac{4}{5}$ of the time steps $\mathcal{B}$ was able to catch $\mathcal{A}$ and keep up with its movements.

In stage 3 sensor data is available so $\mathcal{B}$ is ought not to follow a target position but to maximize its proximity to the other agent by following the sensory gradient. This sensor gradient is only available in stage 3, where the sensor data are part of the net's output. To enable motor inference with sensor gradients, the loss is calculated as the difference of the predicted sensor inputs and the maximal possible sensor inputs for each sensor:

$$\begin{aligned} s_{pref} &:= (1, 1, ..., 1) \\ \mathcal{L}^t &:= \text{MSE}(s^t, s_{goal}) \end{aligned} \tag{5.1}$$

The gradients are calculated accordingly and then applied to optimize the motor commands in order to gain higher sensory activity. Thus the agent follows the gradient of maximal sensory input. The SCV must be disabled for this scenario because this would create obstacle avoidance behavior.
Unfortunately $\mathcal{B}$ did not succeed following $\mathcal{A}$ in under these circumstances. The

direction of maximal sensor input was hardly relevant for the trajectory. The first assumption was that this is due to the target $s_{pref} = (1, 1, ..., 1)$. Trying to maximize the activities of each sensor might lead to gradients where the real wished direction is only marginally included. So the calculation of the preferred sensor stimulus is edited: let sensor $k$ have the highest activity in the sensor prediction.

$$s_{pref}[i] := \begin{cases} 1.0 & \text{if } i = k \\ 0.0 & \text{otherwise} \end{cases} \tag{5.2}$$

Afterwards the point spread function is applied to $s_{pref}$ leading to a preferred sensor activity with the shape of a real sensor activity.
However this extension was also not successful. After further investigations the reason for this was found. As shown in figure 5.10 even though $\mathcal{A}$ heads towards $\mathcal{B}$ , the predicted sensor activity decreases. That means that the model has not sufficiently learned the relationship between motor commands and their resulting perceptual consequences. Thus the sensor gradients could not be propagated back to the motor commands in a meaningful way.

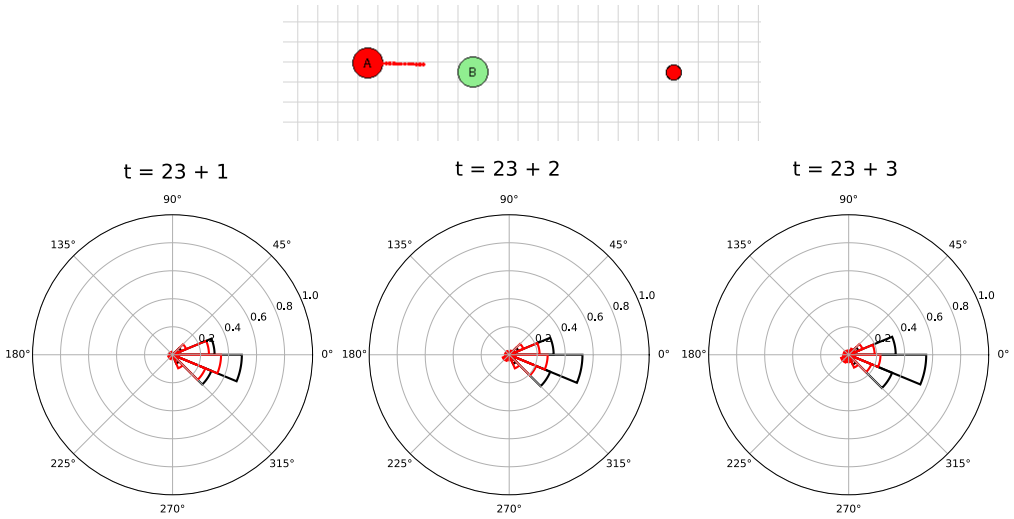


**Figure 5.10:** Sensor predictions (red) and simulated sensor input (black) for $\mathcal{A}$ in chasing task

This might be a consequence of the learning procedure. Due to the agent's unsystematic actions, proximity to obstacles might not be stable in time or fluctuate in their direction and activation. A solution might be to design other learning condition, training with more systematic trajectories.

# Chapter 6

# Conclusion and Future Work

Why are we not surprised by our own movements? An internal model must exist, that is able to predict the results of certain motor commands to our muscular actuators, using the current state of our body and the environment. Fortunately this model is not only useful to prepare ourselves for our own movements, but it can also be used for inverse anticipatory behavior. To reach a certain goal position in space, the motor commands are adjusted and their effect predicted until the anticipated effects match the goal. This active inference process, in which free energy is reduced can only work if the model has been trained sufficiently. Learning follows the same principle of prediction error minimization: predict the outcome of motor commands and adapt the model to minimize the prediction error. Both processes, learning and motor inference were implemented in stage 1. As expected due to experience from previous work (Otte et al., 2017, 2018; Stoll, 2019) the agent was able to learn the properties of its dynamical system quickly and sufficiently to show goal-directed behavior. This could be shown in the active inference task of stage 1, where the agent's performance in moving to a target position was evaluated. The agent did well in this task but performed badly in tasks where obstacles were involved because collisions could not yet be predicted.

In order to interact with the world, an agent not only needs to be able to predict its own dynamics but also perceive surrounding objects and other agents. This capability is crucial to perform interactive behavior like avoiding obstacles or following other agents. The approach in this thesis followed the embodiment idea that the agent should not be a passive observer, detached from the perceiving environment and having symbolic representations of other agents. Instead, the model of sensor dynamics should be learned the same way as the movement dynamics, by predicting future perceptions and adapting the model in order to reduce the prediction error. This still fits into the picture of active inference and the free energy principle, because the desired state is not restricted to physical space. It can be a preferred state in the world or of the agent, which includes a preferred configuration of the agent's joints and body parts as in stage 1, but it can also mean a favored sensory input.

Before implementing sensor predictions, intermediate stage 2 was inserted, which already resolved the lack of obstacle perception by adding sensor input to the model. This additional information enabled the model to refine its predictions and take into account approaching obstacles and borders. It did however not yet anticipate future perception and instead took accurate physically simulated sensor data as input. The results of the performed experiments can thus serve as benchmark for the behavior the agent would express in stage 3 if its sensor predictions were 100% accurate. Two additions were made to the processing of sensor input. First, to prevent signals from unpredictably jumping between sensors, the signal was convolved with a point spread function. This distributed the incoming proximity value to adjacent sensors and enabled a more continuous perception. Secondly, to prevent a random sensor prediction onset the predicted signal was suppressed until a real perception was made. This addition was necessary to prevent the net from predicting obstacles out of nowhere.
As expected, the agent performed well in active inference and obstacle avoidance tasks but only with external assistance. Sensory counter vectors were induced to move the targets away from the direction of perceived obstacle proximity. In addition the individual sensors were weighted by a sensitivity value calculated from the current direction of movement and the direction of the target relative to the agent. These additions to the model were needed to obtain good obstacle avoidance performance in active inference tasks. However in some tasks the SCV prevented the agent to reach its goal because it was in a line with an obstacle and the agent was not able to decide for a direction to pass it. This situation might sound familiar from everyday life where you are not sure on which side to pass a lantern on the street.
In stage 3 the sensor data used for the anticipation of future states were not simulated but predicted by the model. This was learned as described above, by adapting the model to reduce the prediction error. Both the own motor dynamics and the sensor inputs could be predicted well, which enabled the agent to perform active inference tasks. Now instead of using simulated sensor data in each time step of anticipation, the agent uses its own predictions of perception. Experiments showed that the predictions were accurate enough to use the SCV in obstacle avoidance behavior. As expected the performance in reaching the target and avoiding collisions was not as good as the benchmark in stage 2, but better than stage 1 where no sensor data was available.
The prediction of sensor data allowed for an additional possibility: not only a preferred position could now be targeted by reducing the prediction error, but also a preferred sensor input. The sensor gradient was shown to be equally powerful as the SCV in avoiding obstacles. Following a gradient of own predictions is a more internal mechanics than inducing an SCV externally. Additionally the sensor gradient has the advantage of being available even when there is no positional target that should be approached. SCV on the contrary always needs a target position to

be calculated.
Additional experiments were performed to investigate more applications of the sensor gradient. The task to follow an agent by following the gradient of maximal sensor input was unsuccessful because the network was not able to propagate the sensor gradients back to the motor commands in a meaningful way.
Overall the neural network was able to learn to anticipate sensor inputs and use these predictions to actively infer its motor commands in different tasks. Good performance was shown in obstacle avoidance tasks, whereas an exclusive motor control by following sensor gradients is not possible yet.

One challenge that occurred during the experiments was the perception of borders. An agent is not able to distinguish between the perception of another obstacle and of a border. Both induce undistinguishable sensor proximity. This leads to problems when the agent is instructed to follow another agent by following its gradient of maximal sensor input. The agent will eventually fly to a corner because there the overall sensor signal is maximal. The same problem occurs while performing goal-directed behavior with collision avoidance. A goal close to a border is never reached because the agent's collision avoidance mechanics prevent it. To solve this problem, a distinction between a border sensor signal and an agent sensor signal can be implemented. However to stay within the framework of embodiment, this should not be solved via symbolic representations. Instead, an additional quality of perception could be implemented (like a color or shape signal) which enables the agent to learn the distinction of sensor signals. This corresponds to the biological mechanisms of fast perception. A frog's visual system is able to decide whether a perception corresponds to food or a predator by detecting simple visual features: a large expanding area of activation means danger while a quickly moving small area of activation corresponds to a fly (Butz & Kutter, 2016, p.62).

A possible extension to the model in the field of social cognition is to implement ideas of simulation theories. As mentioned in the introduction, simulation theories claim that every observed behavior is simulated by our own brain and thus by the internal sensorimotor model (Barsalou, 2008). This is opposed to the idea of passive observation working with symbolic representations. The agents in this thesis are not detached from the observed environment in this disembodied sense but they are also not yet performing internal simulations of observed behavior. To implement this, the sensor input needs to be mapped to relative position changes, hence velocities since they are the input of the forward model. Then the observed velocities can be simulated by the own internal model to anticipate foreign movements. In fact this also enables to infer the goal of foreign agents. Furthermore in a next step not only the foreign velocities can be anticipated, but also the sensor inputs that the foreign agent might encounter in the future. This allows for mutual social interaction like '*can the other agent see me?*'.

Going back to the active inference principle, another extension of the learning procedure is possible. Currently learning is done by predicting the effect of random motor commands and adapting the model to reduce the prediction error. However the exploration-exploitation idea suggests that learning can be optimized by performing motor commands that lead to trajectories where the predictions of effects are very uncertain. This way the agent decides for itself what it should learn by performing the actions with high uncertainties. This is a similar concept as the philosophical idea of object affordance, which means according to Gibson (2014) "what [the environment] offers the animal, what it provides and furnishes, either for good or ill". An agent must be able to interact with relevant objects, objects that contain relevant information. Robotics use this idea in active vision, that is the agent decides which area of the visual space it should observe to minimize its need for information (Aloimonos, Weiss & Bandyopadhyay, 1988).

A last idea of possible extensions to this thesis is to implement event cognition. The theory of event coding (Hommel, Müsseler, Aschersleben & Prinz, 2001) suggests that both perceptions, actions and again their effects are cognitively structured as common events codes. The event segmentation theory (Zacks, Speer, Swallow, Braver & Reynolds, 2007) additionally defines the start and end points of events. Butz (2016) suggests that these theories can be combined with predictive encodings. A surprise in the prediction signal marks an event boundary. This enables to build hierarchical event structures and plannable chains of actions. Butz, Bilkey, Knott und Otte (2018) implemented a retrospective and prospective inference model, on parts of which this thesis is based. In this approach retrospective inference successfully finds event boundaries which help the agent to perform goal directed behavior, based on the current event. This idea can also be used for social cognition to infer foreign event codes. If the observed agent suddenly changes its trajectory, it may follow a new goal. This surprise in observed behavior can be used to learn and use event structures for other agents in the environment.

The thesis showed that is is possible to learn the anticipation of perceptual effects of action and use it for goal directed behavior. Additionally the predictions can be used to show simple interactive behavior like avoiding other agents. The underlying mechanics are completely grounded on sensorimotor dynamics and thereby follow the idea of embodiment. The model developed in this thesis can be extended with the suggested ideas to refine the sensor gradient mechanics and further investigate the ideas of active inference, event coding and embodied cognition.

# Appendix A

# Algorithms

**Algorithm 1:** Structure of learning procedure in stage 1

```
resetEnvironment()
resetLstmState()
/* Define a time period for the forward pass sequence       */
seqLength = 15
T_from = 0
T_to = T_from + seqLength
foreach forward pass sequence do
    m^t = createRandomMotorCommands()
    /* Perform forward pass to predict next system state    */
    (v^{t+1}, a^{t+1}) = forwardPass((v^t_sim, a^t_sim, m^t))
    /* Simulate motor commands to obtain targets            */
    (v^{t+1}, a^{t+1}) = simulate((v^t_sim, a^t_sim, m^t))
    /* Calculate the loss, the gradients and apply them     */
    L_{v,a} = MSE((v^{t+1}, a^{t+1}), (v^{t+1}_sim, a^{t+1}_sim))
    optimizeWeights(L_{v,a})
    /* Update the time period for the forward pass sequence */
    T_from = T_to
    T_to = T_from + seqLength
end
```

**Algorithm 2:** Structure of motor inference procedure in stage 1

```
resetEnvironment()
resetLstmState()
/* Define a time period for the forward pass sequence */
seqLength = 15
T_from = 0
T_to = T_from + seqLength
P^0 ← (0, 1)
G^0 ← createTargetPosition()
m^0 = m^1 = ... = m^seqLength = (0, 0, 0, 0)
foreach motor inference iteration do
    for t from T_from to T_to do
        /* Perform forward pass to predict next system state */
        (v^t, a^t, m^t, σ^t) --fwPass--> (v^t+1, a^t+1, σ^t+1)
        /* Calculate the target */
        P^t+1 ← P^t + v^t
        v_goal^t+1 ← clip((G − P^t+1), 0.3)
        /* Calculate the loss, the gradients and apply them */
        L_G ← MSE(v^t+1, v_goal^t+1)
        m^t ← optimizeWeights(L_G)
    end
end
/* Perform a real step with the optimized motor commands */
t ← T_from
(v^t, a^t, m^t, σ^t) --fwPass--> (v^t+1, a^t+1, σ^t+1)
/* Update the time period for the forward pass sequence */
T_from ← T_from + 1
T_to ← T_to + 1
```

# Appendix B

# Extension of Stage 3: Working Sensor Gradient

In stage 3 the future sensor inputs are predicted by the forward model. This enables the calculation of a sensor loss $\mathcal{L}_s$, that is the difference between the predicted sensor data $s^{t+1}$ and a defined preferred sensor input $s_{pref}$. The calculated sensor gradient was successfully used to adapt the motor commands $m^t$ in order to avoid obstacles. This was done by combining the sensor loss $\mathcal{L}_s$ with the goal position loss $\mathcal{L}_G$ (see figure4.22). However it was not possible to control an agent by using the sensor gradient alone. The reason is assumed to be a too weak connection between motor commands and effects in perception. The learning procedure did not sufficiently train the correlation of motor commands to their resulting sensor activation. By implication this means that the model does not know how to adapt the motor commands in order to obtain a preferred sensory stimulus.

The proposed solution in the thesis was to improve the learning procedure. However a simple extension of the model fixes the problem without changing the learning procedure. The hypothesis is that the previous model can be interpreted as follows: First calculate the sensor error. Secondly, propagate the error back to the motor commands that are responsible for it. Finally, adapt the motor commands. This implies that the connection "*which motor commands result in which sensor inputs?*" is learned sufficiently. However the model rather learns the two connections "*which motor commands lead to which velocities?*" and "*which velocities lead to which sensor inputs?*". So the inference process can be divided in two components: first the sensor loss is used to calculate to infer the velocities that would be needed to obtain the preferred sensor inputs. Next these goal velocities are used as targets for the motor inference process. This is visualized in figure B.1.

This extension was implemented and tested with the task of the chasing experiment in section 5.3. Agent $\mathcal{B}$ chases agent $\mathcal{A}$ by striving for maximal sensor activation at the sensors in the direction of the currently strongest activation. The performance of $\mathcal{B}$ is evaluated by the distance to the chased agent $\mathcal{A}$ . $\mathcal{A}$ meanwhile approaches a target position that changes every 50 time steps. The results

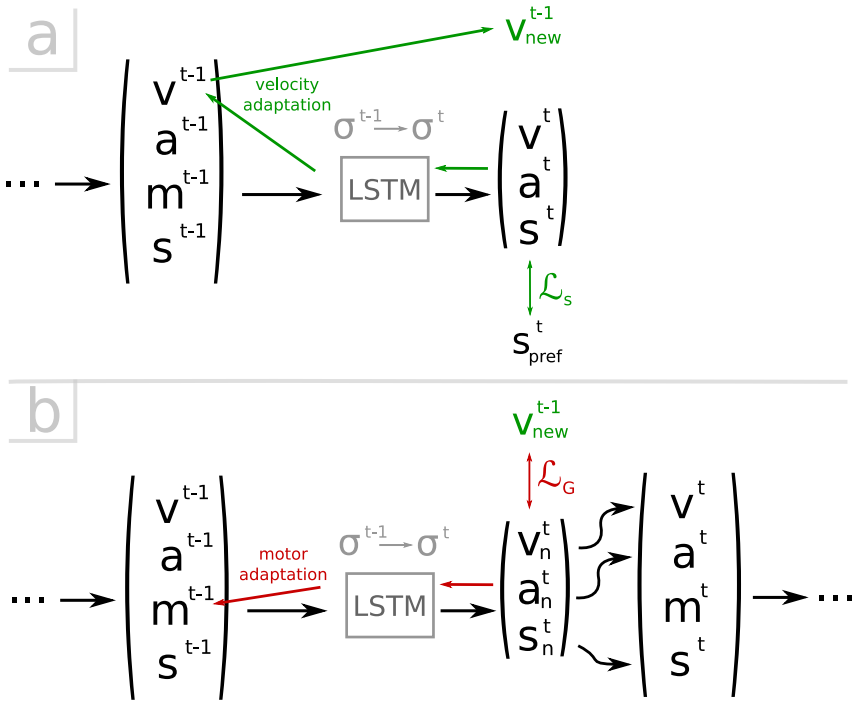


**Figure B.1:** Extension of stage 3. **a)** The sensor gradients are calculated using sensor loss $\mathcal{L}_s$. They are applied to the velocity $v^{t-1}$ to obtain $v_n^{t-1}$ which can be interpreted as the velocity that is required to gain the preferred sensor input. **b)** This goal velocity $v_n^{t-1}$ now serves as target to calculate the velocity loss $\mathcal{L}_G$ and adapt the motor commands.

(visualized in figure B.2) can be compared to the results of the best-case scenario in stage 1 (figure 5.9). In stage 1, $\mathcal{B}$ was able to stay in close range (0.3) to $\mathcal{A}$ in 818 of 1000 time steps, which is the best case scenario. In this extended version of stage 3, $\mathcal{B}$ was able to be in this range in 379 time steps, which shows that the idea was successfully implemented.

By observing $\mathcal{B}$ it can be noted that a change of direction is done slow in relation to $\mathcal{A}$ 's movements. This is probably because the sensor predictions are only accurate in the near future. Thus $\mathcal{B}$ can not plan far ahead. A second observation is that $\mathcal{B}$ is lost at the point where $\mathcal{A}$ is too far away to be detected by the distance sensors. To fix this problem, the sensor distance was set to reach the whole environment. However a low sensor activity can still be hidden in the random noise of sensor predictions, so the chasing behavior is harder with greater distance between $\mathcal{A}$ and $\mathcal{B}$ .
The results show that this extension enables the model to use the sensor gradient for movement control.

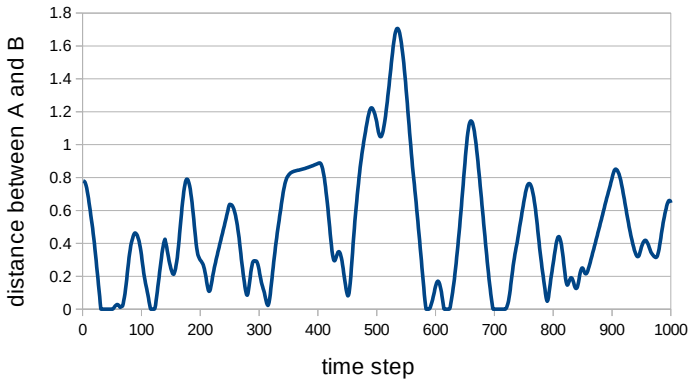


**Figure B.2:** Chasing performance in extended stage 3.

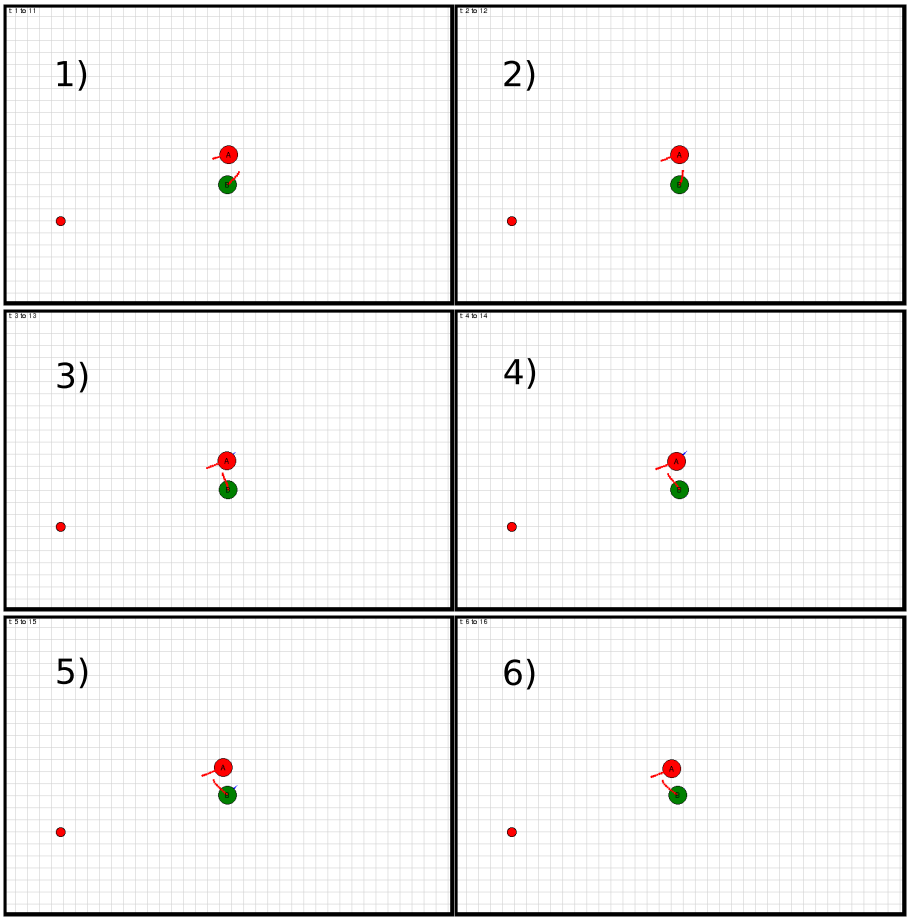


**Figure B.3:** Selected time steps of chasing behavior. The red line depicts $\mathcal{B}$ 's predicted effects of its motor commands. $\mathcal{B}$ adapts its motor commands correctly to follow the sensor gradient.

# Abbreviations

| | |
|---|---|
| BPTT | Backpropagation through time |
| GOFAI | Good old Fashioned Artificial Intelligence |
| LSTM | Long Short Term Memory |
| MDP | Markov Decision Process |
| MSE | Mean Squared Error |
| NN | Artificial Neural Network |
| POMDP | Partially Observable Markov Decision Process |
| RNN | Recurrent Neural Network |
| SCV | Sensor-induced Counter Vector |

# Bibliography


Aloimonos, J., Weiss, I. & Bandyopadhyay, A. (1988). Active vision. *International journal of computer vision*, *1* (4), 333–356.

Astrom, K. J. (1965). Optimal control of markov processes with incomplete state information. *Journal of mathematical analysis and applications*, *10* (1), 174–205.

Barsalou, L. W. (2008). Grounded cognition. *Annu. Rev. Psychol.*, *59*, 617–645.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Butz, M. V. (2016). Toward a unified sub-symbolic computational theory of cognition. *Frontiers in psychology*, *7*, 925.

Butz, M. V., Bilkey, D., Knott, A. & Otte, S. (2018). Reprise: a retrospective and prospective inference scheme. In *40th annual meeting of the cognitive science society*.

Butz, M. V. & Kutter, E. F. (2016). *How the mind comes into being: Introducing cognitive science from a functional and computational perspective*. Oxford University Press.

Di Paolo, E. A. & De Jaegher, H. (2012). The interactive brain hypothesis. *Frontiers in human neuroscience*, *6*, 163.

Dominey, P. F., Prescott, T. J., Bohg, J., Engel, A. K., Gallagher, S., Heed, T., ... Schwartz, A. (2016). Implications of action-oriented paradigm shifts. In A. K. Engel, K. J. Friston & D. Kragic (Hrsg.), *The pragmatic turn: Toward action-oriented views in cognitive science* (Bd. 18, S. 333–355). MIT Press.

Engel, A. K., Friston, K. J. & Kragic, D. (2016). Where's the action? In A. K. Engel, K. J. Friston & D. Kragic (Hrsg.), *The pragmatic turn: Toward action-oriented views in cognitive science* (Bd. 18, S. 1–15). MIT Press.

Fodor, J. (1979). *Representations: Essays on the foundations of cognitive science*. Cambridge, MA: MIT Press.

Fodor, J. A. (1985). Precis of the modularity of mind. *Behavioral and brain sciences*, *8* (1), 1–5.

Friston, K. (2016). *Lstm forward and backward pass*. `http://serious-science.org/free-energy-principle-7602`. Serious Science. (Accessed: 2019-08-25)

Friston, K. J. (2016). The mindful filter. In A. K. Engel, K. J. Friston & D. Kragic (Hrsg.), *The pragmatic turn: Toward action-oriented views in cognitive science* (Bd. 18, S. 97–107). MIT Press.

Friston, K. J. (2017). *Active inference and artificial curiosity.* `https://youtu.be/Y1egnoCWgUg`. University of Edinburgh. (Accessed: 2019-08-27)

Friston, K. J., Samothrakis, S. & Montague, R. (2012). Active inference and agency: optimal control without cost functions. *Biological cybernetics*, *106* (8-9), 523–541.

Gibson, J. J. (2014). *The ecological approach to visual perception: classic edition.* Psychology Press.

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9* (8), 1735–1780.

Hommel, B., Müsseler, J., Aschersleben, G. & Prinz, W. (2001). The theory of event coding (tec): A framework for perception and action planning. *Behavioral and brain sciences*, *24* (5), 849–878.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural networks*, *1* (4), 295–307.

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, *1* (4), 541–551.

Mallya, A. (2017). *Lstm forward and backward pass.* `http://arunmallya.github.io/writeups/nn/lstm/index.html`. (Accessed: 2019-08-24)

McCulloch, W. S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5* (4), 115–133.

Otte, S., Hofmaier, L. & Butz, M. V. (2018). Integrative collision avoidance within rnn-driven many-joint robot arms. In *International conference on artificial neural networks* (S. 748–758).

Otte, S., Schmitt, T., Friston, K. & Butz, M. V. (2017). Inferring adaptive goal-directed behavior within recurrent neural networks. In *International conference on artificial neural networks* (S. 227–235).

Pfeifer, R. & Bongard, J. (2006). *How the body shapes the way we think: a new view of intelligence.* MIT press.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Rumelhart, D. E., Hinton, G. E., Williams, R. J. et al. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, *5* (3), 1.

Stoll, J. (2019). *Rnn-driven rocket ball motion planning in a dynamically changing environment* (Unveröffentlichte Diplomarbeit). Eberhard Karls University Tübingen.

Weisstein, E. W. (2019a). *Circle-line intersection.* `http://mathworld.wolfram.com/Circle-LineIntersection.html`. From MathWorld–A Wolfram Web Resource. (Accessed: 2019-07-11)

Weisstein, E. W. (2019b). *Line-line intersection.* `http://mathworld.wolfram.com/Line-LineIntersection.html`. From MathWorld–A Wolfram Web Resource. (Accessed: 2019-07-11)

Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, *1* (4), 339–356.

Zacks, J. M., Speer, N. K., Swallow, K. M., Braver, T. S. & Reynolds, J. R. (2007). Event perception: a mind-brain perspective. *Psychological bulletin*, *133* (2), 273.

# Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Bachelorarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

| | |
|---|---|
| Ort, Datum | Unterschrift |