



## **D4.6 Context Sensitive Agent Adaption Enabling**



<b>Grant Agreement nr</b>	856879
<b>Project acronym</b>	PRESENT
<b>Project start date (duration)</b>	September 1st 2019 (36 months)
<b>Document due:</b>	28/02/2022
<b>Actual delivery date</b>	28/02/2022
<b>Leader</b>	University of Augsburg (UAu)
<b>Reply to</b>	<a href="mailto:florian.lingenfelser@informatik.uni-augsburg.de">florian.lingenfelser@informatik.uni-augsburg.de</a> <a href="mailto:silvan.mertes@informatik.uni-augsburg.de">silvan.mertes@informatik.uni-augsburg.de</a> <a href="mailto:thomas.kiderle@informatik.uni-augsburg.de">thomas.kiderle@informatik.uni-augsburg.de</a>
<b>Document status</b>	Submission Version

**Project funded by H2020 from the European Commission**

<b>Project ref. no.</b>	856879
<b>Project acronym</b>	PRESENT
<b>Project full title</b>	Photoreal <b>RE</b> altime <b>S</b> entient <b>ENT</b> ity
<b>Document name</b>	Context Sensitive Agent Adaption Enabling
<b>Security (distribution level)</b>	Public
<b>Contractual date of delivery</b>	28/02/2022
<b>Actual date of delivery</b>	28/02/2022
<b>Deliverable name</b>	D4.6 Context Sensitive Agent Adaption Enabling
<b>Type</b>	Report
<b>Status &amp; version</b>	Submission Version
<b>Number of pages</b>	37
<b>WP / Task responsible</b>	University of Augsburg (UAU)
<b>Other contributors</b>	-
<b>Author(s)</b>	Thomas Kiderle, Florian Lingenfelser, Silvan Mertes, Klaus Weber, Elisabeth André
<b>EC Project Officer</b>	Ms. Diana MJASCHKOVA-PASCUAL Diana.MJASCHKOVA-PASCUAL@ec.europa.eu
<b>Abstract</b>	Development of the final version for a Socially-Aware Reinforcement Learning simulation as well as a live system.
<b>Keywords</b>	socially-aware reinforcement learning, backchannel feedback, personality adaption
<b>Sent to peer reviewer</b>	Yes
<b>Peer review completed</b>	Yes
<b>Circulated to partners</b>	No
<b>Read by partners</b>	No
<b>Mgt. Board approval</b>	No

## Document History

Version and date	Reason for Change
v1.0 10.01.2022	Document created by Thomas Kiderle
1.1 14-02-2022	Version for internal review (14 days before submission date)
1.2 28-02-2022	Revisions in response to review: final version submitted to Commission

## Table of Contents

<b>1. GLOSSARY</b>	<b>4</b>
<b>2. EXECUTIVE SUMMARY</b>	<b>4</b>
<b>3. BACKGROUND</b>	<b>5</b>
<b>4. INTRODUCTION</b>	<b>5</b>
<b>5. BACKGROUND</b>	<b>6</b>
5.1. Backchannel Feedback	6
5.2. Socially-Aware Reinforcement Learning	6
<b>6. CONTEXT SENSITIVE AGENT ADAPTION ENABLING</b>	<b>8</b>
6.1. Reinforcement Learning System	8
6.1.1. Q-Learning and K-Armed Bandit	10
6.1.2. Linear Function Approximation	11
6.1.2.1. TDC	12
6.1.2.2. Feature extraction	13
6.1.2.3. Off-Policy Control using linear function approximation	13
6.2. Context Sensitive Personality Adaption	17
6.2.1. Background	17
6.2.2. Modelling	19
6.2.3. Agent behaviour enabling	22
6.3. Simulation Results	24
6.3.1. Bandit	25
6.3.2. Q-Learning	26
6.3.3. TDC	27
<b>7. CONCLUSION</b>	<b>29</b>
<b>8. REFERENCES</b>	<b>30</b>
<b>9. APPENDIX</b>	<b>31</b>

## 1 GLOSSARY

Term	Definition/Explanation
MDP	Markov-Decision-Process is a class of decision problems, for which only the current state is relevant for the decision making
TDC	"Linear temporal difference with gradient correction" is a reinforcement learning algorithm for linear function approximation
GTD2	"Gradient temporal difference" is a reinforcement learning algorithm for linear function approximation
BML	Behavior-Markup-Language which is used as a protocol for controlling the nonverbal behavior of virtual agents
SSML	Speech-Synthesis-Markup-Language which is used as a protocol for controlling the speech characteristics/content for TTS systems
TTS system	System, which converts Text-to-Speech

## 2 EXECUTIVE SUMMARY

The deliverable 4.6 "Context Sensitive Agent Adaption Enabling" (R, PU, M30) presents the final Socially-Aware Reinforcement Learning system. Since the PRESENT agent aims at establishing engaging interactions, the adaption process is required to be as unobtrusive as possible. Therefore, we use our social sensing pipeline for enabling context sensitiveness and Socially-Aware Reinforcement Learning to adapt the agent. To establish additional engagement and naturalness through empathy, the feedback of the reinforcement learning is further processed by our emotion calculation system.

In this deliverable, we developed the final version for a Socially-Aware Reinforcement Learning simulation as well as a live system. The system currently adapts the backchannel behaviour to the personality preferences of the user and on this basis outputs how intensely the emotions are expressed. Using our adapted version for linear function approximation, the simulation modus is used to find the best parameter configuration for our live system. We also show how this adapted backchannel behaviour is then applied in an embodied agent.

### 3 BACKGROUND

This deliverable contributes to the workpackage WP4 “Behavioural Sensitivity and Responsiveness”. For the user-focused adaption to be enabled, we relied on the previous work done in deliverable 4.1, 4.4 “Non-verbal Agent Behavior Enabling” for the behaviour part and deliverable 4.5 “Agent Social Interpretation Enabling” for using our sensing system to establish user-focused context. In this deliverable, we report on the final version of our context-sensitive agent adaption system, which allows an unobtrusive and user-focused adaption to behavioural aspects in the human-agent interaction. This includes a continuously learning live system and the corresponding final simulation system. To be able to learn continuously, we describe how we adapt linear function approximation methods. Additionally, we present the implemented reinforcement learning model for the backchannel style adaption used to establish rapport with the user and show how the context sensitive adaption is enabled in an embodied agent. We also report on experiments using our simulation, which are used to determine efficient parameters for the final live learning process.

### 4 INTRODUCTION

Establishing rapport between the agent and the user requires an appropriate application of the corresponding non-verbal behaviour (e.g. showing coordination, positiveness). However, this kind of behaviour can sometimes be complex (e.g. applying an appropriate backchanneling in the right situation), and scripting these behaviours would be very time-consuming. Also, the agent doesn’t know which kind of interaction the user individually prefers (e.g. extroverted vs introverted behaviour). The agent can only find out by interacting with the user, which is the typical case for applying reinforcement learning. Traditional approaches require mostly explicit feedback, which is not desirable for agent-human interaction. By giving explicit human feedback (e.g. pushing a button for liking/disliking), it can be tiring for the user in the long run and even in the short run destroy the immersion with the agent. Hence, we aim at retrieving a more unobtrusive way to gather feedback from the user, which is satisfied by our Socially-Aware Reinforcement Learning approach. There we gather feedback from social signals and learn appropriate behaviour.

In this deliverable, we will establish rapport with the user by combining personality cues for extraversion with backchannel feedback to determine a backchannel feedback style. The resulting different styles and our social sensing pipeline will then be used to adapt unobtrusively the backchannel style of the agent to the user preferences.

The subsequent sections of the deliverable are structured in the following way. Section 4 introduces the main concepts on which our implementation is based. These encompass Backchannel Feedback and the approach of Socially-Aware Reinforcement Learning.

Section 5 introduces the algorithms used for implementing the user-focused adapting live and simulation system. These encompass the Bandit, Q-Learning and with a special focus on linear function approximation. Therein, we show how the function approximation has to be adapted to be able to learn in a continuously ongoing, nonstationary and nonlinear behaving environment. The output of this chapter describes an algorithm that has been implemented by our live and simulation system.

Section 6 first establishes the connection between personality adaption, where extraversion cues are used to express the corresponding personality (discussed in detail in deliverable 4.1), and backchannel feedback aiming at establishing rapport with the user. This background is followed by presenting our reinforcement learning model for adapting the backchannel style to the user preferences. Subsequently, we show how the adapted behaviour is enabled with an embodied agent.

Section 7 presents the experiments that were conducted in our simulation system to find the best performing algorithm in combination with the best parameter configuration. The result serves as input for our live system.

Section 8 concludes the deliverable.

## **5 BACKGROUND**

In this deliverable we will show how we implemented the user-focused personality adaption of the agent with respect to social signals as interactional context. For adapting the personality, we concentrated on the learning task of applying backchannel feedback appropriately in accordance with the preferred personality. Therefore, we will first do a short recap on the psychological background of backchannel feedback. Since we used reinforcement learning for the user-focused adaption, another section concludes our Socially-Aware Reinforcement Learning approach. Both will be necessary to understand the implementation treated in the residual deliverable.

### **5.1 Backchannel Feedback**

---

Human conversation uses backchannel feedback as an unobtrusive listener response addressing the content or situation of the interaction, which serves among others communicational aims like showing attention (e.g. smiling) or acknowledgement (e.g. nodding). Superficially, backchannel feedback can be categorized into verbal and nonverbal types. The verbal backchannels can be among others categorized as its interactive function. Since we only refer to reactive expressions in this deliverable, our focus is on this verbal type. These occur often after a speaker's turn and serve as a reaction to the spoken content. Which of these reactive expressions were implemented for the backchannel style adaption can be looked up in section 6.2.1.

The non-verbal backchannel feedback is subdivided by the used modalities. Applying gazing and smiling explicitly for backchanneling purposes has not been used in this deliverable. Nodding and head shakes can either be used in isolation or combined with verbal backchannel types. Additionally, the application is not restricted to a single usage, also multiple follow-up applications are possible according to the literature. Further details on the theory of backchannel behaviour can be looked up in deliverable 4.4 "Non-verbal Agent Behavior Enabling".

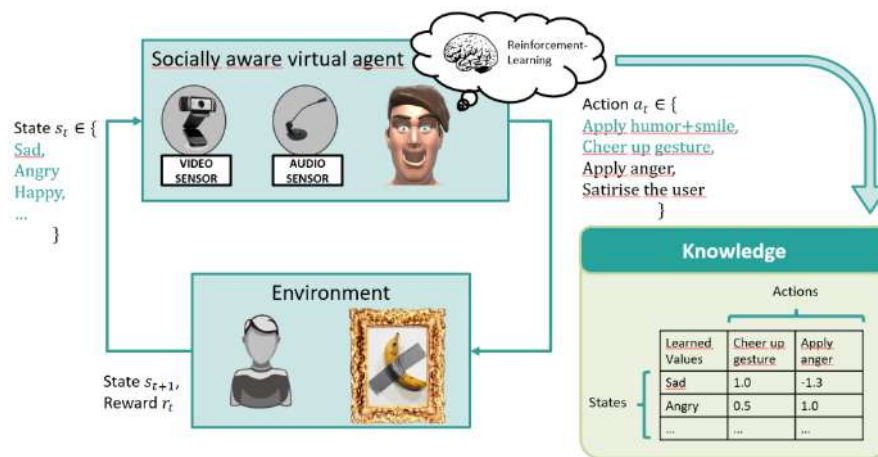
### **5.2 Socially-Aware Reinforcement Learning**

---

The basic idea of reinforcement learning is that an autonomous system (also called an agent) learns stepwise via trial and error to take the correct actions in certain situations. In order to make decisions, the agent must perceive the environment [1, p. 4] by abstracting it as states with features relevant to the application. Choosing and

executing an action leads to the next step, where a new state and a reward for the action execution is observed. This reward will be accounted for with the knowledge available at this time step. The agent's overall goal is to maximize the reward [1, p. 48] in the long run, for which it is accumulating a state-action value for each state-action pair. For most approaches, these "learned" values are combined with a strategy (e.g. always choosing actions with the high values) and serve as a basis to identify appropriate actions in the corresponding situation. Therefore the agent has to find a good tradeoff between choosing the highest value (exploitation of the knowledge) and choosing randomly (exploration).

For interactions between humans and agents, it is not sufficient to request reward explicitly from the user (e.g. keystroke ratings). This can become tiring over time, destroy the immersion with the user, and hence bias the requested value. So it is rather desired to get the information more unobtrusively by accounting social signals. This can either refer to explicitly tracked values (e.g. laughter, gestures) or to values combined from multiple tracked signals (e.g. engagement, emotions). Similarly, state information can be retrieved by social signals and by accounting its features abstracted to a state. Extending the basic ideas of reinforcement learning with unobtrusively requested social information results in maintaining the immersion during human-agent interactions. This approach is called Socially-Aware-Reinforcement Learning. An example for this modified learning method is shown in figure 1.



**Figure 1:** Exemplary overview of Socially-Aware Reinforcement Learning

The example shows a scenario, where the user interacts with an agent. In each timestep, the tracked user emotion is encoded to the agent's state. Based on this emotion and its current knowledge, the agent learns to mitigate bad emotions using appropriate mitigation strategies encoded into the action space. Another example would be a personality adaption, which we will address in this deliverable. More information on the basic ideas of reinforcement learning or the Socially-Aware Reinforcement Learning approach can be looked up in Deliverable 4.1.



## 6 CONTEXT SENSITIVE AGENT ADAPTION ENABLING

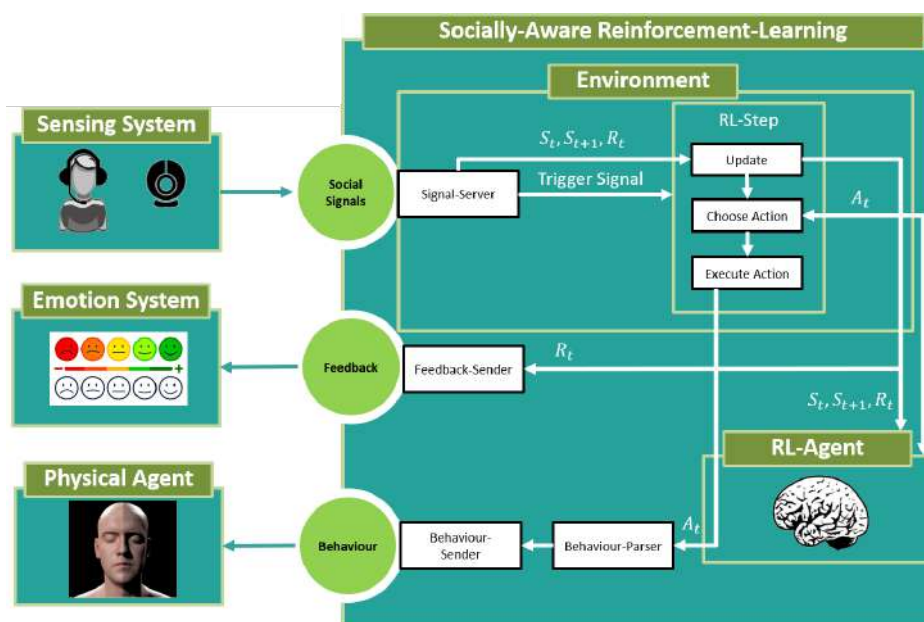
To enable context-sensitive agent adaption, we implemented a system realizing our Socially-Aware-Reinforcement Learning approach. In the next section, we will give an overview of the implementation of this system, where we outline the architecture of the live simulation system. Additionally, the working of each system and the realized algorithms will be described. Another section will explain, how we establish rapport through the personality adaption modifying the backchannel style, how the learning problem has been modelled, how the behaviour actually has been enabled through our live system and how well the implemented algorithms have been performed in simulation experiments.

### 6.1 Reinforcement Learning System

To be able to adapt the agent unobtrusively to the user's needs we implemented a system for Socially-Aware Reinforcement Learning. This system can be used online for live interactions or for simulation purposes. Both modes will be presented in the following. Afterwards, we will present the different algorithms which we implemented for our reinforcement learning system. The Socially-Aware scenario requires capabilities to react to changes in the  $q^*$  values because of user preference changes and therefore continuous learning. Hence, our algorithms and configurations are selected to also address the nonstationary behaviour of the environment. Additionally, it is known that off-policy algorithms yield better results. Therefore, we restrict our implementation to this class of reinforcement learning algorithms.

#### Live Modus:

The architecture of online learning is shown in figure 2.



**Figure 2:** Socially-Aware Reinforcement Learning live system architecture

Our Socially-Aware Reinforcement Learning system is one component of our overall pipeline. To be able to consider social signals we rely on the social sensing system, which already has been developed and described in Deliverable 4.5. Since our learning



success will influence the agent affectively, we also included our emotion calculation system, which has been described in Deliverable 4.4. In order to show an effect of the personality adaption during the interaction, additionally a physical agent is used. The exact realization with the agent is explained in section 6.2.3. To be independent from other systems, we implemented the Reinforcement Learning system in a distributed way, meaning that it communicates with all other components via the network. In the following, we describe how the live and the simulation modus of our Socially-Aware Reinforcement Learning system works.

The social sensing system is configured to output among others social signals, that are defined by the learning task and the corresponding reinforcement learning modelling. For the modelling in this deliverable, we use the valence value of the last user turn. Therefore, we average all valence values from the user turn to one value on the sensing system side, which is then sent to our Socially-Aware Reinforcement Learning system.

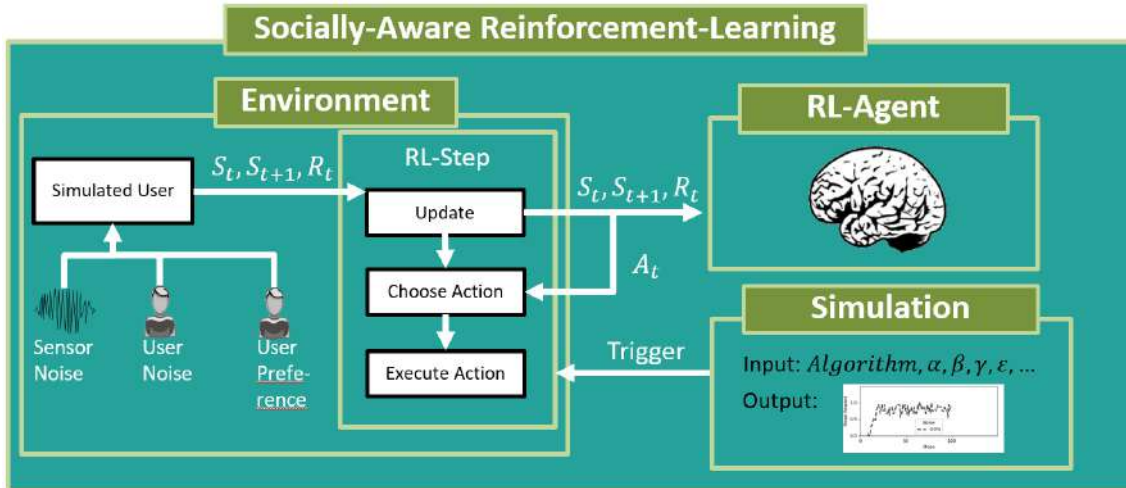
On the RL System side, we receive all social signals from the social sensing system that are relevant for the learning. This can range from continuously sent signals to single values as the averaged valence value. One or more of the signals serve as a trigger signal, which triggers one reinforcement learning step (in our case the single valence value). For our learning task in this deliverable the averaged valence value serve as a trigger signal and the emotions calculated from our emotion calculating system is also sent to the RL system.

Within one triggered learning step, the agent usually infers a state, and a reward from the social signals (see section 6.2.2 for our modeling). Because of our modeling it is not necessary to assemble the reward. The calculated reward represents the current learning success of the agent and is passed to a network component. The latter one sends the feedback to the emotion calculation system, where the reward is accounted to emotions. The reward and state are additionally used to update the agent's knowledge. Afterwards, the next action is chosen and executed by translating it to a protocol which the agent supports (e.g. BML). The translated action is passed to another network component, which sends commands to the physical agent (see section 6.2.3 for more details). Then the physical agent then actually executes the action. Once a new trigger signal is sent, the Socially-Aware Reinforcement Learning system is idled until the next trigger signal.

### **Simulation Modus:**

The simulation modus of our system omits any network component previously mentioned. Hence, there is no connection to the outside for either the social sensing system, emotion calculation system, or the physical agent. So the system doesn't wait for a trigger signal and runs directly with the specified input parameters. These encompass, among others, the learning algorithm,  $\alpha$  (learning rate), (in case of gradients)  $\beta$  (second learning rate),  $\gamma$  (discounting factor),  $\epsilon$  (exploration rate), the number of episodes and steps. The episodes are equal to the number of different users that will be simulated by one run. For each user, a random preference, sensor noise, and user noise will be initialised. The sensor noise is used to simulate a nondeterministic behaviour of the sensor for measuring the user's social signals, which leads to a bias in the rewards calculation. Similarly, humans tend to not behave the same facing the same situation. So we also simulate this nondeterminism through the specified user noise value, which likewise introduces a bias in the feedback. These values altogether will contribute in calculating the reward in each step. A concrete

instance from our experiments will be shown in section 6.2.2. Additionally, humans can change preferences during the interaction. This can be simulated by specifying timesteps, where each user will change its preference to another random one during the simulation. This will show, how fast the algorithm will “recover” from the preference change. The architecture for our simulation is illustrated in figure 3.



**Figure 3:** Socially-Aware Reinforcement Learning simulation system architecture

When the simulation has terminated, a graph for the simulation is plotted showing the average reward for each step. This yields the performance of the corresponding algorithm with respect to the passed parameters.

### 6.1.1 Q-Learning and K-Armed Bandit

We implemented the K-Armed Bandit, because in our scenario a simple modeling already may be sufficient and in this case, the agent can benefit from this lightweight approach. It abstracts from the full reinforcement learning approach in terms of states. Hence, the agent can not distinguish different situations and approximates an averaged value for each available action. Addressing our requirements of nonstationarity and in contrast to the classic bandit, we chose the learning rates  $\alpha \in [0, 1)$  for the bandit to be constant. This way more recent values are weighted higher than values gained at more distant timesteps. The classical algorithm (without the modification of  $\alpha$ ), that has been implemented, can be looked up in Sutton and Barto [1]. It can be derived from the classical Q-Learning algorithm, when only one empty state occurs, the learning rate averages the formula using an action counter for the updated action (e.g.

$\frac{1}{n(action)}$  ) and  $\gamma = 0$  is chosen. To also enable state-based modeling with a tabular based approach, this traditional Q-Learning algorithm is also implemented illustrated in figure 4.

**Initialize:**  $Q(s, a) = 0 \forall s \in \mathcal{S} \forall a \in \mathcal{A}(s)$ , Initial state  $s$   
**for all steps**  $t = 0, 1, 2, \dots$  **do**  
 $a \leftarrow \begin{cases} \arg \max_{a'} Q(s, a'), & \text{if } rand \geq \epsilon \\ \text{random } a & , \text{if } rand < \epsilon \end{cases}$   
 apply  $a$ , measure next state  $s'$  and reward  $r = \mathcal{R}(s, a, s_{t+1})$   
 $Q(s, a) \leftarrow Q(s, a) + \alpha \left[ r + \gamma * \max_{a'} Q(s', a') - Q(s, a) \right]$   
 $s \leftarrow s'$   
**end for**

**Figure 4:** Q-Learning Algorithm

Similar to the Bandit implementation, we chose a constant  $\alpha$ , to also address the nonstationary learning problem in case of including states.

### 6.1.2 Linear Function Approximation

Tabular reinforcement learning methods (like the ones previously introduced) are appropriate for learning problems with a limited number of states. However, some reinforcement learning problems have a very large or even an infinite state space (e.g. infinite Markov-Decision-Problem (MDP)), where an infinite number of state-action values would have to be learned using these tabular methods. Linear function approximation addresses this problem by learning an infinite amount of states concurrently with a finite effort. Additionally, some features in the state space can be related to others (as in our use case the expressivity level of different features). Hence, extending linear function approximation with a feature extraction method (e.g. Fourier base as discussed later) enables learning also nonlinear dependencies among the features despite using linear approximation methods.

Considering finite MDP's the state value  $V(s) \forall s \in \mathcal{S}$  is calculated directly. However, regarding infinite MDPs this would take a very long time, possibly even infinitely long, since states could be entered infinitely often that had never been visited before, or states that had already been visited could only reappear after infinite time. This would change the behaviour of the agent only very slowly or not at all. In the context of reinforcement learning, the idea of function approximation is to find an approximation  $\mathcal{V}_{\vec{\omega}}$  for the state value function  $\mathcal{V}_{\pi}$ . Thereby, a finite parameter vector  $\vec{\omega}$  - also called approximator - is used to approximate the state values  $\mathcal{V}_{\vec{\omega}}(s)$  for all  $s \in \mathcal{S}$ , so that  $\mathcal{V}_{\vec{\omega}}(s) \approx \mathcal{V}_{\pi}(s)$  (see [2], [3], and [1]). Hence, the approximator  $\vec{\omega}$  determines implicitly the shape of the approximated function  $\mathcal{V}_{\vec{\omega}}$ . [2]–[5]

For simplicity, solely the approximator for the state value function will be discussed at first. All explained concepts can be applied equivalently to the action value function, which is why the corresponding algorithm is discussed only in section 5.1.2.6. The overall learning goal is to minimize the error between the real state value  $\mathcal{V}_{\pi}(s)$  and the approximated value  $\mathcal{V}_{\vec{\omega}}(s)$  for all states. To achieve this, the gradient method (also called steepest descent/ascent method) is used, since it is a meanwhile established approach for the optimization problem with linear function approximation. A gradient is a mathematical operator, which differentiates a function according to multiple variables

concurrently. This enables to investigate the descend/ascent of a function in multiple directions at once. Using this operator on an error measure can be used to search a local minimum or maximum of a function regarding this measure.

Detailed derivations of the gradient and the algorithms would exceed the amount of this deliverable, since the gradient always disappears through the derivation for considered algorithms. So we reference chapter 9 of Sutton and Barto [1] for more details.

Since off-policy (using different strategies for learning and generating behaviour) algorithms usually perform better than on-policy approaches (using the same strategy for both), we also restricted our selection of the linear function approximation to this kind of methods. However, it is a nontrivial problem to use off-policy function approximation in combination with bootstrapping (using approximations of other state action pairs during the value update) and still guarantee the algorithm to converge against the real values. Therefore, Sutton and Barto [1] suggest to minimise the error between  $\mathcal{V}_\pi(s)$  and  $\mathcal{V}_{\tilde{\omega}}(s)$  by calculating the gradient of the Projected-Bellman-Error (PBE). The algorithms presented in the next two chapters will minimize this error measure.

We implemented both gradient-based algorithms. To additionally apply a Fourier base for learning dependencies among the features we present it in another section. In the subsequent sections, we explain the adaption of both to be able to implement an algorithm, that approximates the action-value function and converges.

#### 6.1.2.1 TDC

Figure 6 shows an off-policy prediction algorithm for linear function approximation and is similarly derived as the GTD2 algorithm. The parameter and working are also similar to the GTD2. Since the performance seems to be better according to the literature [6], we focus our experiments on the TDC algorithm. Every update of  $\vec{\omega}$  applies as follows:

```

Initialize:  $\gamma, \alpha, \varepsilon, \vec{\omega} \leftarrow \vec{0}, \vec{\theta} \leftarrow \vec{0}, s$ 
procedure TDC( $s, s', r, \vec{\theta}, \vec{\omega}$ )
     $\delta = r + \gamma \vec{\phi}^T(s') \vec{\omega} - \vec{\phi}(s)^T \vec{\omega}$                                  $\triangleright$  compute delta
     $\vec{\omega} \leftarrow \vec{\omega} + \alpha * \delta \vec{\phi}(s) - \alpha * \gamma \vec{\phi}(s') * (\vec{\phi}(s)^T \vec{\theta})$          $\triangleright$  update  $\vec{\omega}$ 
     $\vec{\theta} \leftarrow \vec{\theta} + \beta * (\delta - \vec{\phi}(s)^T \vec{\theta}) \vec{\phi}(s)$                          $\triangleright$  update  $\vec{\theta}$ 
    return  $(\vec{\omega}, \vec{\theta})$ 
end procedure
    
```

**Figure 6:** TDC Algorithm

First,  $\delta$  for correcting the error is computed. Next to learning the vector  $\vec{\omega}$  another vector  $\vec{\theta}$  is learned using a second learning rate  $\beta$  and accounting  $\delta$ . Because of  $\vec{\omega}$ 's dependency on  $\vec{\theta}$  it is necessary for the convergence of the algorithm, that  $\beta > \alpha$  applies. One step before updating  $\vec{\theta}$ , this second vector is used to update the vector  $\vec{\omega}$ , which we are actually interested in.

As described in Busoniu and Sutton et al. [2]–[6], linear function approximation is characterized by the linear combination of the parameter vector  $\vec{\omega}$  with the features

used to describe a state. Features are represented by a vector  $\vec{\phi}$ , where  $\phi_i : \mathcal{S} \rightarrow \mathbb{R}$  applies and  $\dim \vec{\omega} = \dim \vec{\phi}$  must hold. Resulting from this definition,  $\mathcal{V}_{\vec{\omega}}(s)$  can be calculated as follows:

$$\mathcal{V}_{\vec{\omega}}(s) := \vec{\phi}(s)^T \vec{\omega} = \sum_{i=1}^n \vec{\phi}_i(s) \vec{\omega}_i, \forall s \in \mathcal{S}$$

#### 6.1.2.2 Feature extraction

To also approximate non-linear functions and therefore, learn dependencies between several features from the state, the literature proposes feature extraction methods (e.g. Coarse Coding, Tile Coding, Polynomials, Fourier basis). We decided to use the Fourier basis, since it seems to fit our use case and it has been shown by [4], [5] despite its easy usage to be a powerful method. With the help of the Fourier basis every function can be represented as a sum of trigonometric functions. In principle, this corresponds to a linearization of the function, since parameters and features are still multiplied and added linearly. It depends on the degree of transformation how exact the approximation is. Although there exist also a univariate Fourier base, we decided to use the Fourier base of multivariate functions to be more flexible in learning. In contrast to univariate functions, these are functions, which depend on more than one variable [4], [5]. The Fourier base of multivariate functions is defined as follows:

#### Definition 1 (n-th Fourier base):

Let there be a single, normalized feature vector  $\vec{x} = (x_1, \dots, x_m)^T$  with  $x_j \in [0, 1]$ , which represents a unique fix state  $s \in \mathcal{S}$ . Furthermore, let there be a set of coefficient vectors  $C = \{\vec{c}_1, \dots, \vec{c}_k\}$  with  $k = (n + 1)^m$ . Then the Fourier basis of the n-th order is defined by  $\phi_i(s) = \cos(\pi \vec{c}_i^T \vec{x})$ , whereas  $\vec{c}_i = (c_1, \dots, c_m)^T$  using  $c_j \in [0, \dots, n]$ ,  $1 \leq j \leq m$  and  $1 \leq i \leq k$ .

When combining the Fourier basis with a linear function approximation algorithm, the Fourier basis is usually calculated for an input state, which outputs a new feature vector representing the Fourier basis. This will then serve as the new feature vector, which then serves as input to the corresponding algorithm.

#### 6.1.2.3 Off-Policy Control using linear function approximation

To be able to learn an action value function using the TDC and GTD2, several adaptations have to be made regarding  $\alpha$ ,  $\phi$  and the Fourier base calculation. All will be introduced subsequently and finally the resulting algorithm will be presented. For simplicity reasons only the modification of TDC will be shown. The GTD2 adaption works accordingly.

#### Adaption of $\alpha$ and $\phi$ :

It is known from the literature [6] that the usage of a decreasing  $\alpha$  for linear function approximation leads to convergence for GTD2 and TDC. However, we want to continuously learn and therefore use a constant learning rate. So several assumptions have to be determined for these algorithms to converge. First experiments have shown, that too big values for  $\alpha$  and individual  $\phi_i$ 's lead to divergence of the



algorithm, because the resulting  $\delta$  is increasing and alternating. This leads to overcorrecting the weights in the opposite direction for every update and in the end to an alternating divergence.

One reason is, that even with an appropriately chosen  $\alpha$ , each  $\phi_i$  influences also the size of the update. Hence, for convergence each  $\phi_i$  should not be truly bigger than 1 or truly smaller than -1. This can be achieved either by the modeling of the problem itself or an intermediate step, which scales all  $\phi_i \in \mathbb{R}$  to a value  $\phi_i \in [0, 1]$ . For our implementation we decided to directly encode all features to values between 0 and 1. For the usage of the Fourier basis this even is a crucial input condition. The feature vector output of the Fourier basis contains always values restricted to  $\phi_i \in [-1, 1]$ , so the convergence condition remains satisfied.

Another reason for this is that the algorithms don't consider the size of the feature vector. The bigger the size of the feature vector becomes through feature selection (e.g. Fourier basis or manually), the smaller the individual weights  $\omega_i$  have to be to lead to the same state value. This can be restricted by choosing  $\alpha \in [0, \frac{1}{k}]$ , whereas  $k$  is the size of  $\phi$ .

### **Adaption of Fourier base:**

Using all possible coefficients for a Fourier base of the  $n$ -th order, will be in most cases too exhaustive. So we introduced in our algorithm the possibility to restrict these coefficients by a coupling parameter. The coupling determines how many values in each coefficient vectors are allowed to be unequal to 0. Hence, using coupling of 1 and an order of 2, a vector like (0,1,1) or (2,1,0) are not allowed, whereas vectors like (1,0,0) or (0,0,2) are included.

Until now the Fourier basis has been defined for state-describing feature vectors. In general, however, the feature vector should be able to differentiate actions in each state to be able to learn for each action how good it is in the current state. So we have extended the Fourier base. The state action attribution remains unique learning for every action an own state-parameter vector mapping. We have defined this as follows:

### **Definition 2 (Action differentiating Fourier base):**

Let  $\vec{\phi}(s)$  be a  $k$ -dimensional feature vector which encodes an arbitrary state  $s \in \mathcal{S}$  as a Fourier base and let  $\mathcal{A}(s) = \{a_1, \dots, a_n\}$  be a finite set of possible actions. Then the action differentiated Fourier base is defined by:

$$\vec{\phi}(s, a_i) = \begin{pmatrix} \delta_{(a_1)(a_i)} \phi_1(s) \\ \vdots \\ \delta_{(a_1)(a_i)} \phi_k(s) \\ \vdots \\ \delta_{(a_n)(a_i)} \phi_1(s) \\ \vdots \\ \delta_{(a_n)(a_i)} \phi_k(s) \end{pmatrix}, 1 \leq i \leq n$$

### Off-Policy Control algorithm:

As suggested by [6], for our application the parameter  $\beta$  has been set with  $0.5\alpha$  in relation to the learning rate. Subsequently the off-policy control algorithm with the above presented modifications will be presented. We assume that the feature vector  $\phi$  is normalized at each point in time, since otherwise the Fourier basis is not unique anymore. Figure 7 depicts the complete pseudocode for the algorithm, which is subdivided into 3 procedures.

The first is as usual the updating procedure *update*, to which the old state  $s$ , the executed action  $a$ , the new state  $s'$  and the obtained reward  $r$  is passed. Similar to the original algorithm first the delta is calculated. Therefore, the function *qvalue* is used returning the current state action value. At first, the state is transformed to its Fourier base and subsequently, the resulting vector is multiplied with the parameter vector (procedure *qvalue*, line 29-36). The Fourier base is calculated as shown before (procedure *fourier*, line 14-28). The first loop (line 17-19) calculates the state basis according to the predefined coupling, whereas the second loop outputs the action differentiated Fourier basis (line 20-26). Thereby, we assume that the array has been initialized with 0's. The lines 6-12 are nearly equal to the original algorithm. The only exception is that we use the previously generated Fourier base to update the parameter vector  $\vec{\omega}$  and the vector  $\vec{\theta}$  for correcting the error. The procedure *update* is called always after executing an action. The selection of the action is solved with the  $\epsilon$ -greedy policy, which chooses the currently best action to a probability of  $1 - \epsilon$ .



```

Initialize:  $\gamma, \alpha, \varepsilon, \beta \leftarrow \frac{1}{2}\alpha, \omega[], \theta[], s, a, \mathcal{C} = [\vec{c}_1, \dots, \vec{c}_k]$ 
procedure UPDATE( $s, s', a, r$ )
     $\delta = r + \gamma \max_{a'} qvalue(s', a') - qvalue(s, a)$   $\triangleright$  compute delta
     $\varphi_1[] = \text{fourier}(s, a)$   $\triangleright$  get action basis for  $s$ 
     $\varphi_2[] = \text{fourier}(s', \max_{a'} qvalue(s', a'))$   $\triangleright$  get action basis for  $s'$ 
     $\nu = 0$ 
    for  $i \leftarrow 1$  to  $k * \mathcal{A}.length$  do
         $\nu \leftarrow \nu + \varphi_1[i] * \theta[i]$ 
    end for
    for  $i \leftarrow 1$  to  $k * \mathcal{A}.length$  do
         $\omega[i] = \omega[i] + \alpha \delta \varphi_1[i] - \alpha \gamma \varphi_2[i] \nu$   $\triangleright$  update  $\tilde{\omega}$ 
         $\theta[i] = \theta[i] + \beta (\delta - \nu) \varphi_1[i]$   $\triangleright$  update  $\tilde{\theta}$ 
    end for
end procedure
procedure FOURIER( $s, a$ )
     $basis[] = \text{new basis}[k]$   $\triangleright$  array starts with 1
     $basis_{actions}[] = \text{new basis}_{actions}[k * \mathcal{A}.length]$   $\triangleright$  array starts with 1
    for  $i \leftarrow 1$  to  $k$  do
         $basis[i] = \cos(\pi * \vec{\phi}(s)^T \mathcal{C}[i])$   $\triangleright$  compute Fourier basis
    end for
    for all  $a_j \in \mathcal{A}(s)$  do  $\triangleright 0 \leq j < \mathcal{A}.length$ 
        if  $a_j == a$  then
            for  $i \leftarrow 1$  to  $k$  do
                 $basis_{actions}[j * k + i] = basis[i]$ 
            end for
        end if
    end for
    return  $basis_{actions}$ 
end procedure
procedure QVALUE( $s, a$ )
     $basis[] = \text{fourier}(s, a)$ 
     $value = 0$ 
    for  $i \leftarrow 1$  to  $k * \mathcal{A}.length$  do
         $value \leftarrow value + basis[i] * \omega_i$ 
    end for
    return  $value$ 
end procedure

```

**Figure 7:** Off-Policy Control TDC Algorithm

## 6.2 Context Sensitive Personality Adaption

In the following, we will show the relation between personality adaption, backchannel feedback and rapport, on which our context sensitive adaption is founded. Another section will explain the reinforcement learning modeling for the live as well as the simulation modus of our realized application. The subsequent section explains how we implemented the context sensitive agent adaption physically through an embodied agent.

### 6.2.1 Background

One of our overall goals is to establish rapport between the user and the agent. Rapport is usually perceived as a harmonic connection between participants. This is often also described as good interpersonal chemistry or as it has “clicked”. Literature suggests that this psychological phenomena evolves from different nonverbal cues communicated during conversations, which shows at least one of the three columns of rapport:

- **Positiveness** emerges from a positive presence of the interlocutor, either nonverbally (e.g. by smiles, nodding) or verbally using positive speech, which communicates a friendly and caring aura
- **Mutual attentiveness** is connected to showing mutual engagement through nonverbal cues (e.g. turning towards the other interlocutor) or verbally referring to aspects in the conversation
- **Coordination** is achieved by establishing a bidirectional synchronicity with other interlocutors using nonverbal behaviour (e.g. appropriately applied backchannel feedback) [7]

As shown in Deliverable 4.1, personality adaption to the preferences of the user (either similarity or opposite attraction) has an influence on measures like engagement. Therefore, expressing and adapting the personality appropriately also contributes to establishing rapport with the agent. Since extraversion is most salient, our implementation of the personality adaption focuses to this trait. Personality is expressed through cues of different modalities, Figure 8 shows the extraversion cues relevant for our behaviour adaption. Their implementation in our scenario is explained in section 5.2.3.

	Source	Introvert	Extrovert
Gestural	Gesture amplitude	• narrow	• Wide range of movement/broad gestures
	Gesture rate	• Low	• Correlation: High speech rate → high gesture rate • more movements of head, hands and legs
	Gesture speed	• Slow	• Fast, quick
Emotional	Speech/ Facial expressions	• Low emotional expressivity	• High emotional expressivity
Paralinguistic	Speech	• Slow speech rate • Low volume	• High speech rate • High volume
	Conversational speech behaviour	• Less backchannel feedback	• More backchannel feedback • Initiating conversation

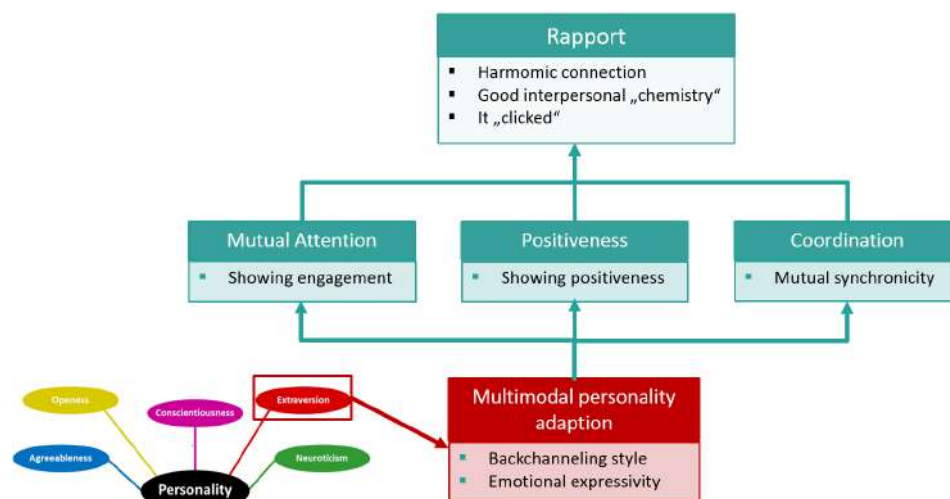
Figure 8: Extraversion cues

To express an adapted personality we combined these cues with backchannel feedback types, which will be used by the agent to show that it is an active listener. Figure 9 shows all relevant backchannel types. The modeling for adapting the backchannel style will be presented in the next chapter, its implementation with an agent will be explained in section 5.2.3.

	Type	Characterisation	Examples
Verbal	Backchannel	Non-lexical audio expressions	"hm", "huh", "oh", "mhm", "uh huh"
	Reactive expression	Short lexical, phrase/word	"oh really/really", "yeah", "gee", "okay", "sure", "exactly", "allright", "man", "shit", "hell"
	Repetition	Repetition of parts of the turn	Speaker: "I buy 110 per year!" Listener: "110"
	Acknowledgement	Acknowledging utterances	"Yeah", "Mm hm"
Nonverbal		Head nodding	<ul style="list-style-type: none"> <li>Nod Standalone</li> <li>Or combined with short verbal backchannel feedback                             <ul style="list-style-type: none"> <li>1-multiple times</li> </ul> </li> </ul>

**Figure 9:** Extraversion cues

As a result, the agent is adapting to the user's personality by modifying its backchannel style with respect to extraversion cues, that match the current preferences and hence, the preferred personality of the agent. By showing an attention, coordination and positiveness through appropriate backchanneling, the agent establishes rapport with the user. Figure 10 graphically illustrates this underlying idea.



**Figure 10:** Establishing rapport through personality adaption using backchannels and emotional expressivity

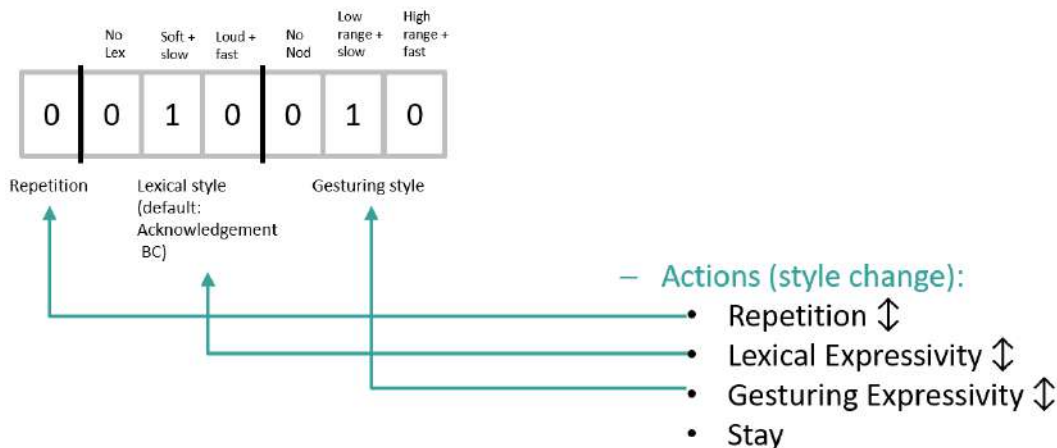
For more information on personality adaption see Deliverable 4.1, more detailed insights on rapport and backchannel feedback can be found in Deliverable 4.4.

### 6.2.2 Modelling

In this section we present the reinforcement learning modelling for the backchannel style adaption of the agent. Several behaviours can be inferred from the state, which will be explained subsequently.

#### States and Actions:

As shown in Figure 11, we modelled the state as a binary vector subdivided into 3 sectors which indicate how to apply backchannels. Regarding the second sector (digit 2-4), the position of the 1 indicates how to apply the lexical backchannel. If it is at the left most position, no lexical backchannel is applied. In the middle the lexical backchannel is uttered slowly and with soft volume, whereas on the right most position the lexical backchannel will be loud and fast. The third section works similarly combining the nodding backchannel with the corresponding extraversion cues regarding gestures (left: no nodding, middle: nodding with low movement range and slow, right: nodding with high movement range and fast). The first section solely consisting of the first digit indicates whether the combination of the other two sections will be applied once (equals 0) or twice (equals 1).



**Figure 11:** States and actions of the reinforcement learning

Since the actions of the physical agent are encoded into the state space, we modelled the reinforcement learning actions to move the 1 of the corresponding section. With respect to the example lexical\_style  $\uparrow$  moves the 1 of the second section to the right. Hence, the original 1 turns to 0 and moves one place to the right. To reduce complexity, actions without effect are not available in the corresponding state. As an example, repetition  $\downarrow$  would be not available in the example state of figure 11. To be able to learn one of the user preferences for a particular backchannel style, we additionally added the action stay, which let the state to remain unchanged. A fully trained agent is desired to choose stay most of the time (e.g. except exploring).

**Live Modus Reward:** To enable a context-sensitive agent adaption using an unobtrusive feedback from the user, we defined the reward to equal the valence of the last user turn. This will be used to rank the last applied backchannel feedback of the agent and measure, whether the emotions of the user after applying the backchannel are good.

**Simulation Reward:** At the begin of the simulation a random user preference is initialized. Assuming this is for example the preference of figure 11 (0 010 010), then a user is simulated, which prefers a soft/slow uttered lexical backchannel combined with a low range/slow nod. Considering the first digit, the whole backchannel combination would be preferred to be applied only once.

Since no social signals are available to assemble the reward, we calculate a base reward, which is then biased to be more realistic. We assume, that in the live mode the user would always show positive valence, if the last action is moving “towards” the preferred state, and shows negative valence otherwise. Hence, we simulate these reactions by calculating a distance from the current state to the actual preference state of the simulated user. In other words, the distance estimates, how many actions through a optimal action selection are required to reach the optimal state.

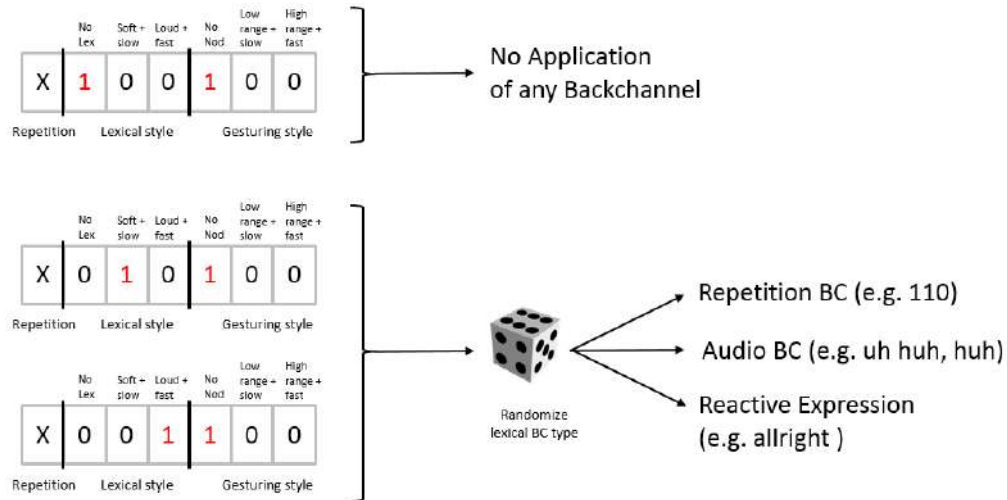
The base reward is calculated by finding out, whether the last action increased the distance to the preference. In this case, a reward of -1 is given. If the distance is decreased, a base reward of 0.5 is returned. If the distance is unchanged, we check whether we stayed in the preferred state. In this case, a base reward of 1.0 is retrieved. Otherwise the agent would stay in a not preferred state, where we give a reward of -0.5. The distance of a state to the preference is calculated by counting how much actions would be needed to reach it, when the agent would have chosen the immediate way to the preference.

Additionally, we have to simulate, that the underlying reinforcement learning task is non-deterministic. On one hand real sensors contain noise to a certain degree, on the other hand humans can also slightly show altered behavior (e.g. through distraction or mood). Hence, we bias the base reward to a realistic reward using the noise probabilities previously presented. According to both of the noise probabilities a random value  $\in [-0.5, 0.5]$  is added to the base reward.

### **Inferred behaviour:**

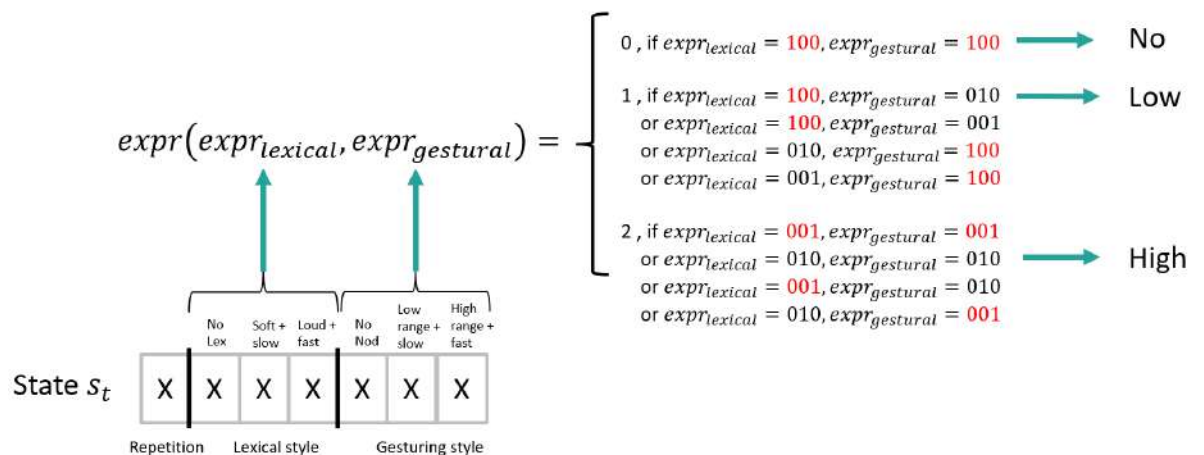
There are a few special cases, where additional behaviour can be inferred for the agent. If no lexical style and gesturing style should be applied, then the agent will omit the backchannel for this turn. Otherwise, if the state brings the agent to omit nodding (1 is at the most left place), but apply a lexical backchannel, then the agent chooses randomly between the other backchannel types previously presented. It can choose between all types except the acknowledgement backchannel. Both special cases are illustrated with figure 12. The X in the state space stands for a wildcard in the state, for which both a 0 and a 1 is possible.





**Figure 12:** Special cases, from which additional behaviour can be inferred

To additionally adapt the emotional expressivity, we also infer how intensely the agent should express the current emotion it retrieves from the emotion calculation system. Therefore we calculate an expressivity level for each timestep by involving the lexical expressivity and the gestural expressivity from our state encoding. Figure 13 shows an overview for every possible configuration. Let the lexical section equal to 001 (the maximum expressivity regarding extraversion) and let the gestural section be also 001 (also the maximum expressivity), then the expressivity function would return a high expressivity level for this step. This resulting expressivity level is then used to trigger either a smooth or an intense emotion (matching the emotion retrieved from the calculation system).



**Figure 13:** Inferring emotional expressivity from the state

The context-sensitive adaption has been introduced in this section through the reinforcement learning model. This context sensitive adaption will be enabled through our implementation in the live system considering the speech, nonverbal gestures and

emotional expressivity. How we enable this behaviour will be shown in the subsequent section.

### 6.2.3 Agent behaviour enabling

To show the progress of the current learning process in terms of adapted backchannel behaviour, our live system has to enable the corresponding behaviour for a certain virtual agent. We decided to implement our system with the UPF Agent, because it supports all important BML commands necessary to realize the nodding and showing emotions with a certain degree of intensity. To be also able to implement different speech expressivity levels, we decided to use the State-of-the-Art TTS System Cerevoice. How we combined them to enable the behaviour adaption, will be shown subsequently.

#### Emotional Expressivity generation

After executing the chosen Reinforcement Learning action to alter the state, we assemble the agent behaviour from the state. First, the emotional expressivity is calculated as presented in the preceding chapter and then further accounted to a float for a BML command. If a value of 0 is returned, the expressivity is set to 0.0. If 1 is returned the expressivity is weak and therefore set to 0.6. If it equals 2, this strong expressivity leads to a value of 1.0. Based on the agent emotion, which is read out by our Emotion Calculation System, and this intensity level for this emotion, a complete BML command uniting both is generated. An example is shown in Figure 14. The additional attribute shift triggers the emotion until another emotion is triggered by our calculation system or the reinforcement learning system. We decided to trigger it over a longer period because otherwise, its perceivability suffers.

```
{
  "type": "behaviours",
  "data": [
    {
      "type": "faceEmotion",
      "emotion": "HAPPINESS",
      "start": 0,
      "attackPeak": 0.25,
      "relax": 0.75,
      "end": 1,
      "amount": 1.0,
      "shift": True
    }
  ]
}
```

**Figure 14:** BML command for triggering happiness

#### Nodding style generation

Depending on the state entries, an additional JSON entry for the data array is generated. No nodding would be previously caught by the special case, where no JSON entry is added. If the state indicates low nodding expressiveness a low expressive nodding is generated, otherwise a high expressive version. Both can be looked up in figure 15.



### Low expressivess

```
{
  "type": "head",
  "lexeme": "NOD",
  "start": 0,
  "ready": 0.3,
  "strokeStart": 0.3,
  "stroke": 1,
  "strokeEnd": 1.7,
  "relax": 1.7,
  "end": 2.1,
  "amount": 0.2
}
```

### High expressivess

```
{
  "type": "head",
  "lexeme": "NOD",
  "start": 0,
  "ready": 0.3,
  "strokeStart": 0.3,
  "stroke": 0.9,
  "strokeEnd": 1.3,
  "relax": 1.5,
  "end": 1.8,
  "amount": 0.6
}
```

Figure 15: Left:low expressive noddng, Right: high expressive noddng

## Linguistic style generation

In the special case that no nodding is applied, it is chosen to generate one of the repetition, audio and reactive backchannels. Subsequently, for the selected backchannel one instance is also randomly chosen and then embedded into a SSML command. This command depends again on the expressivity, where the rate attribute implements the speech rate and the volume attribute realizes the audio volume of the speaker uttering the corresponding backchannel. This SSML command is then passed to Cerevoice generating a wav file with this backchannel and prosodic expressivity. Figure 16 gives an overview for this process and the different instances of repetition, audio and reactive backchannels.

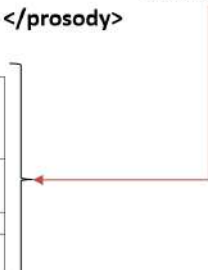
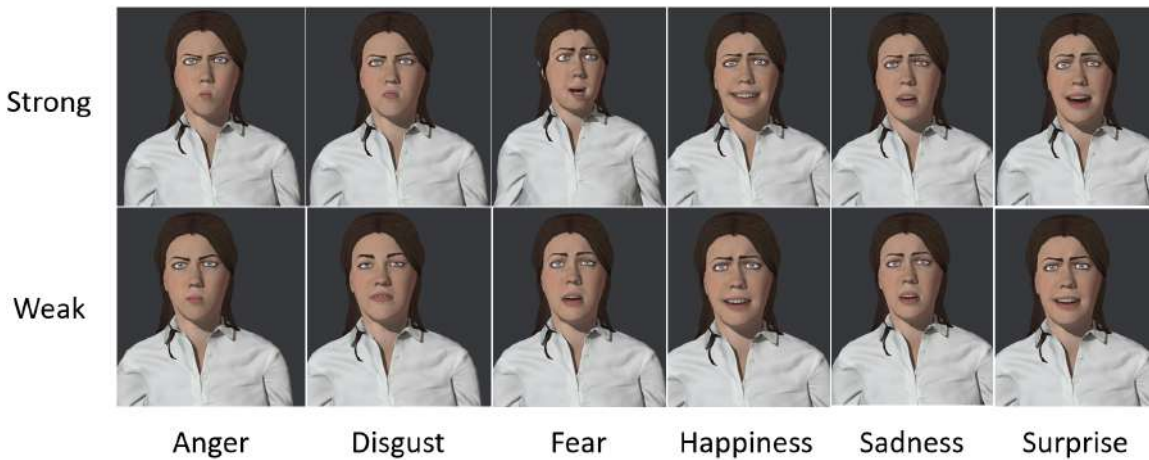
Low expressivess		High expressivess	
<b>&lt;prosody rate="x-slow" volume="silent"&gt;</b>		<b>&lt;prosody rate="x-fast" volume="loud"&gt;</b>	
BACKCHANNEL		BACKCHANNEL	
<b>&lt;/prosody&gt;</b>		<b>&lt;/prosody&gt;</b>	
reactive backchannel	'okay' 'alright' 'right' 'I see'		
audio backchannel	'<spurt audio="g0001_014"/>', # hm 'ah <prosody speed="fast"> <emphasis>huh</emphasis></prosody>' # uh huh		
repetition	repeating last user noun		
acknowledgement	yes yeah		

Figure 16: SSML for low and high expressive backchannels

This figure also shows the instances for the acknowledgement backchannel, which is only applied if nodding is present.

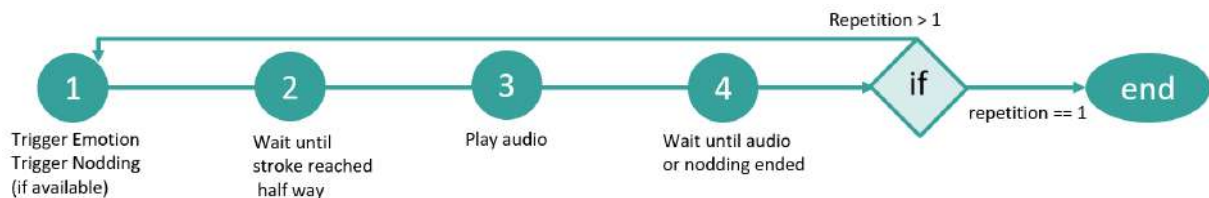
## Enabling the agent behaviour

After generating emotions, nodding and linguistic backchannels, we enable this behaviour by triggering the corresponding behaviours. First the emotion and if available the nodding is triggered by sending the BML command to the agent. In case of emotions one of the 6 basic emotions anger, disgust, fear, happiness, surprise or sadness is triggered. If no emotion has been available a neutral face will be triggered. Each of the basic emotions with the corresponding weak and strong instances are depicted in figure 17.



**Figure 17:** Each of the 6 emotions with its weak and strong form

In case of the nodding, either the weak or strong form is triggered. To align the speech with the nonverbal backchanneling, we wait until the nodding reached the half way of the stroke phasis. Subsequently the audio backchannel (if available) is played with the corresponding prosodic expressivity cues. After that, the program waits until the audio and nodding both ended. Depending on the repetition entry in the state, this procedure either repeats at step 1 or terminates. In case of terminating, the agent will idle, until the next trigger signal for the next reinforcement learning step will be sent. This whole procedure is visualized by figure 18.



**Figure 18:** Procedure of behaviour enabling

### 6.3 Simulation Results

To enable the context-based backchannel style adaption most efficiently in the live modus of our application, we aimed to find the best configuration in terms of algorithm type and corresponding parameters for our simulated agent. Figure 18 shows all parameter configurations used.

We addressed the tradeoff between learning fast, where  $\alpha$  should be not too big, and learning despite precisely on the long term, hence  $\alpha$  also shouldn't be too big. So we chose it to be in a range of  $\alpha \in [0.1, 0.2, 0.3]$ . The gradient is a special case, because according to literature  $\alpha$  should be defined in dependence of  $\beta$ . Additionally, both  $\frac{1}{k}$  should be (as described previously) not exceed  $\frac{1}{k}$  (with k as the number of features).

So we chose  $\beta = \frac{1}{k}$  and  $\alpha$  to depend on it with the corresponding factors. Also a tradeoff between exploration and exploitation has to be chosen appropriately, where we defined  $\epsilon \in [0.05, 0.1, 0.2]$ . To investigate how far the agent should consider future rewards, we decided to set gamma to a narrow (0.5), middle (0.7) and wide sight (0.9) for all state based algorithms.

K-Armed Bandit		Q-Learning			TDC		
$\alpha$	$\epsilon$	$\alpha$	$\epsilon$	$\gamma$	$\alpha$	$\epsilon$	$\gamma$
0.10	0.05	0.10	0.05	0.90	0.90 $\beta$	0.05	0.90
0.20	0.10	0.20	0.10	0.70	0.70 $\beta$	0.10	0.70
0.30	0.20	0.30	0.20	0.50	0.50 $\beta$	0.20	0.50

**Figure 18:** Parameter configurations for the experiments

Each possible permutation within one algorithm has been conducted as an experiment, resulting in 9 experiments for the Bandit, 27 experiments for the Q-Learning and 27 experiments for the gradient algorithm.

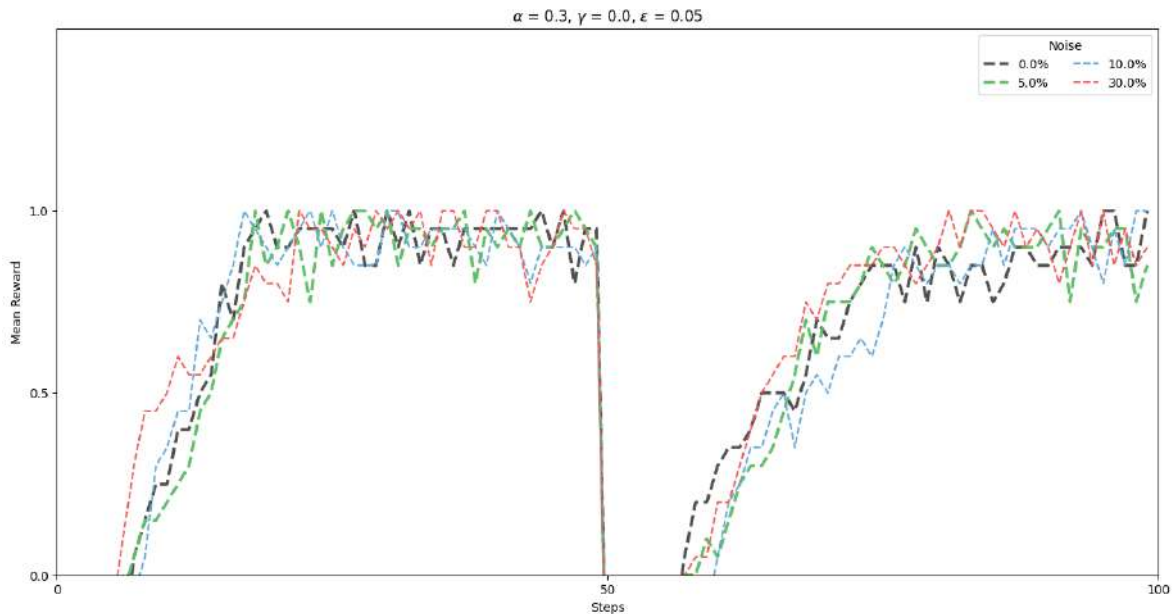
As previously stated, every experiment starts with a random initialization for the preference of the simulated user and is conducted with 4 noise probabilities. These are uniformly assigned for the sensor and the user noise. For both, we introduced the noises to occur never (0.0), rarely (5%, 10%) and frequently (30%). Since our learning environment is nonstationary and therefore is able to change in terms of preference changes, we additionally applied a preference change in the middle of each experiment. This reveals the ability of the configuration to recover from preference changes. As performance measure we chose the average reward, which equals the average reward the agent has obtained in step  $x$  over all episodes. So for each datapoint in our plots we average over all episodes to get the average reward (for an example see the following chapters).

### 6.3.1 Bandit

Since the state based model encodes the actions to be applied by the agent in the state and Bandits lacks states, we encoded each state permutation of the states from the state based agent as one action. This results in e.g. an action derived from "0 001 100" meaning 1 repetition, high expressive linguistic backchanneling and no nodding. In total, the agent chooses between 18 different actions. Using this modelling, we additionally had to adapt our reward computation for the bandit only. If the last action matches the preference +1 is given, otherwise -1. We simulated 100 steps over 30 episodes.

The best configuration is in our opinion the agent with  $\alpha = 0.3, \epsilon = 0.05$ . Although higher learning rate is expected to be less accurate, no such effect has been observed. However, a higher  $\alpha$  (e.g. 0.3) in combination with this exploration rate of 0.05 is performing better at the beginning and also after preference change, than the other

learning rates. The other configurations with  $\epsilon > 0.05$  show a worse performance because of the exploration frequency. Although this can help recovering the agent after a preference change, on the long run the agent performs worse. Figure 19 shows the plot for our preferred configuration.



**Figure 19:** Most performant K-Armed Bandit

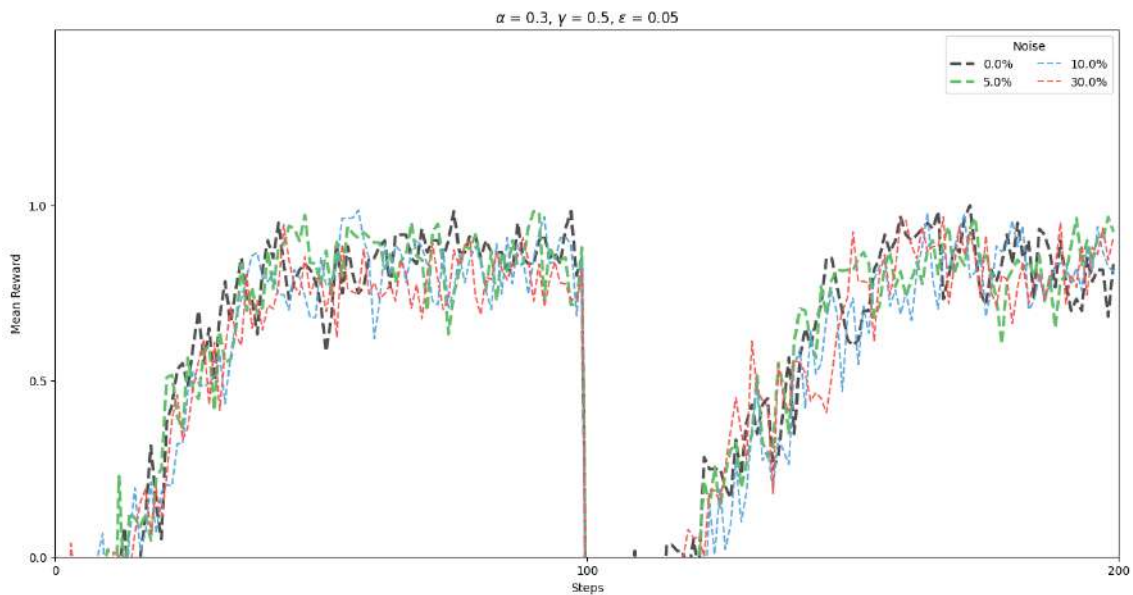
The x axis stands for the steps done for each simulated user. The y axis shows the average reward that has been calculated over all the episodes for the corresponding step. The graph shows that the agent is rapidly adapting to the preference in the beginning and after the preference change. Despite the noise the graph stays relatively stable, although a slight correlation between the amount of noise and the alternating graph can be observed (e.g. 30% noise compared to 10%). The graphs from the other Bandit experiments can be looked up in the appendix.

### 6.3.2 Q-Learning

The Q-Learner is as previously described state based and is using the presented reinforcement learning model. In contrast to the Bandit, the agent is now restricted to alter one style dimension at once. This leads to the advantage of a more unobtrusive behaviour adaption and therefore presumably more consistent behaviour since the agent isn't allowed to jump between 2 backchannel styles altering multiple dimensions anymore (e.g. from "apply no linguistic backchannel and nod strong expressively" to "apply strong expressive linguistic backchannel and no nod"). For the Q-Learning experiments we simulated 200 steps over 30 episodes.

For all experiments it can be observed that (independently of other parameters) the lower the  $\gamma$  is, the better the agent performs especially after a preference change. We assume that this lies in the fact that the future values are higher weighted. However, this future value is based on an approximation, which is temporarily false after a preference change. So the higher the  $\gamma$ , the more the agents accounts these false

approximations and this presumably is the reason why it struggles recovering with high  $\gamma$  values. From these considerations we assume a  $\gamma$  of 0.5 to be the best configuration. Similar to the Bandit, a higher value than  $\epsilon = 0.05$  introduces too much random behaviour. Hence, we consider again  $\epsilon = 0.05$  to be the best value. There can no difference between the learning rates with respect to the accuracy be observed, however the fastest recovering learning and on the long run best performing learning rate is again  $\alpha = 0.3$ . Hence our best configuration is the  $\alpha = 0.3, \gamma = 0.5, \epsilon = 0.05$ . The plot is depicted in figure



**Figure 20:** Most performant configuration for Q-Learning

Similarly to the Bandit, the Q-Learner is relatively stable once it learned the preference. Despite the more consistent behaviour through the state model, there are also downsides with this approach. As can be seen in the graph, first the agent needs more steps to reach a nearly stable behaviour. This also counts for recovering from the preference change. Additionally, noise has a larger impact on the agent than it had for the K-armed Bandit.

Both issues presumably are based in the fact that the agent has to return to the preference state, once he left it (as it is the case for e.g. noise and exploration). Since the positive rewards are only given when reducing the distance, this positive value first has to propagate across the different state-action approximations, before it can take a perceivable effect (in this case returning or staying at the preference state). Hence, choosing between the Q-Learning and K-Armed Bandit for context sensitive agent adaption enabling it has to decided, whether the performance or the consistent behaviour is valued higher. The other Q-Learning experiments can be looked up in the appendix.

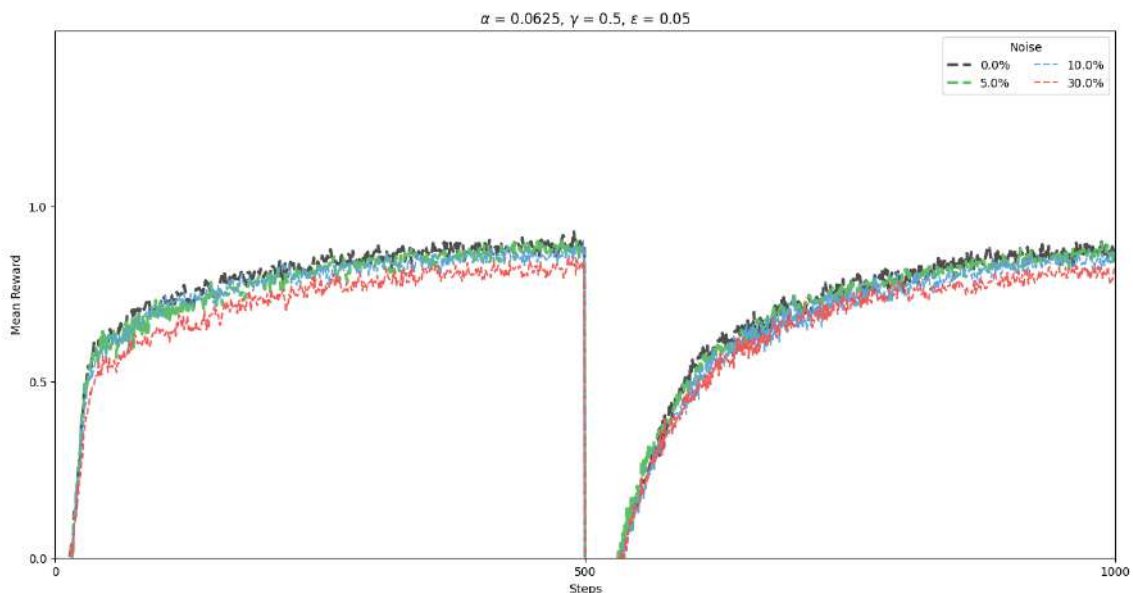
### 6.3.3 TDC

Since the literature (see Sutton et al. [6]) proposed that TDC is more performant than GTD2, we chose this algorithm for our experiments. This algorithm similarly to the Q-Learning agent is state based and therefore expected to introduce a consistent



behaviour. The usage of function approximation introduces additional parameters. Because of convergence issues we chose  $\beta = 1k$ , whereas  $k$  equals the size of the feature vector (as previously described). Our features could be directly the state vector, since its values all equal 0 or 1. However, we decided to apply a fourier base on the state to additionally learn dependencies between the features. Since the agent has to adapt quickly online and presumably only has to approximate a linear function, we chose a simple fourier base of order 1 and a coupling of 1. For nonlinear functions also a more complex base can be chosen. Regarding our experiments we chose 1000 episodes, each having 1000 steps.

Similar to the Q-Learning agent and for the same reasons, we observed a  $\gamma = 0.5$  to be the best and an  $\epsilon = 0.05$ . However in contrast to the other algorithms, the smallest learning rate  $\alpha = 0.5\beta$  is considered to be the best. This is presumably resulting from the function approximation in general, that the adaption concerns all weights and therefore has to be small to not overestimate other values, which are perhaps already well approximated. The resulting plot is depicted by the following figure.



**Figure 21:** Most performant TDC configuration

Overall, the algorithm learns to adapt to the user preferences, but comparatively slowly. Whereas the algorithm relatively quickly learns to adapt in the beginning, it has a slow recovery time after preference change. Another issue is that it's similar to the Q-Learner more prone to noise than the K-armed Bandit. We account for the same reason as for the Q-Learner, that the agent has difficulties in returning to the preference state and staying there (in noise and exploration cases). Beyond this, the linear function approximator has an additional complexity, because it learns all states at once for an action. So it is additionally difficult to find back to the correct preference state.

The strength of TDC is its generalisation capabilities, which can increase accuracy and speed of learning compared to the previously presented algorithms. Although the state dimensions of linguistic and nodding expressivity have potential for generalisation using a fourier base, the overall problem contains less possibilities for learning dependencies. So we assume that this problem may not be complex enough and the algorithm too heavy to take use of the actual capabilities of the gradient based reinforcement learner.

Choosing another fourier base would give rise to an even worse problem, because it is already chosen to be as simple as possible and would just introduce more heaviness to the learning problem. The linear function approximation has more potential than the q-learner to learn a complicated problem, but therefore the learning task needs more underlying generalization capabilities.

## 7 CONCLUSION

In this deliverable, we presented the final version of our Social-Aware Reinforcement Learning system together with its live and simulation mode. After adapting the linear function approximation and fourier base, we conducted experiments to find the best algorithm configuration using the K-Armed Bandit, Q-Learning and TDC algorithm for the backchannel style adaption.

Based on the state-based modelling previously presented, the Q-Learner and TDC algorithm both performed best using  $\gamma = 0.5, \epsilon = 0.05$ , since this way the agent explores but doesn't behave randomly too often. Additionally, as we presumed, the agent has also difficulties because of temporarily false approximations to find its way back to the preferences state after preference changes, where it seemed advantageous to choose a tendentious small  $\gamma$ . For versions without states we directly encoded the different styles as actions. This K-Armed Bandit performed in many ways better than the state based versions.

Despite the longer time for finding the preference, using states also introduced a more consistent behaviour because the agent maximally alters one modality of the backchannel style at once. Which version should be chosen depends on the preferences for the system. If consistent behaviour is more important than the duration for finding the preference we recommend the Q-Learner over the others. This is because it is more lightweight for the learning task at hand than the TDC gradient. For the live system of the PRESENT agent we however rather use the K-Armed Bandit. On one hand it performs better with higher noises, on the other hand it matches the preference faster, which then leads faster to the preferred behaviour. We assume the user not to change his preference too often, so this will also lead relatively fast to a consistent behaviour.

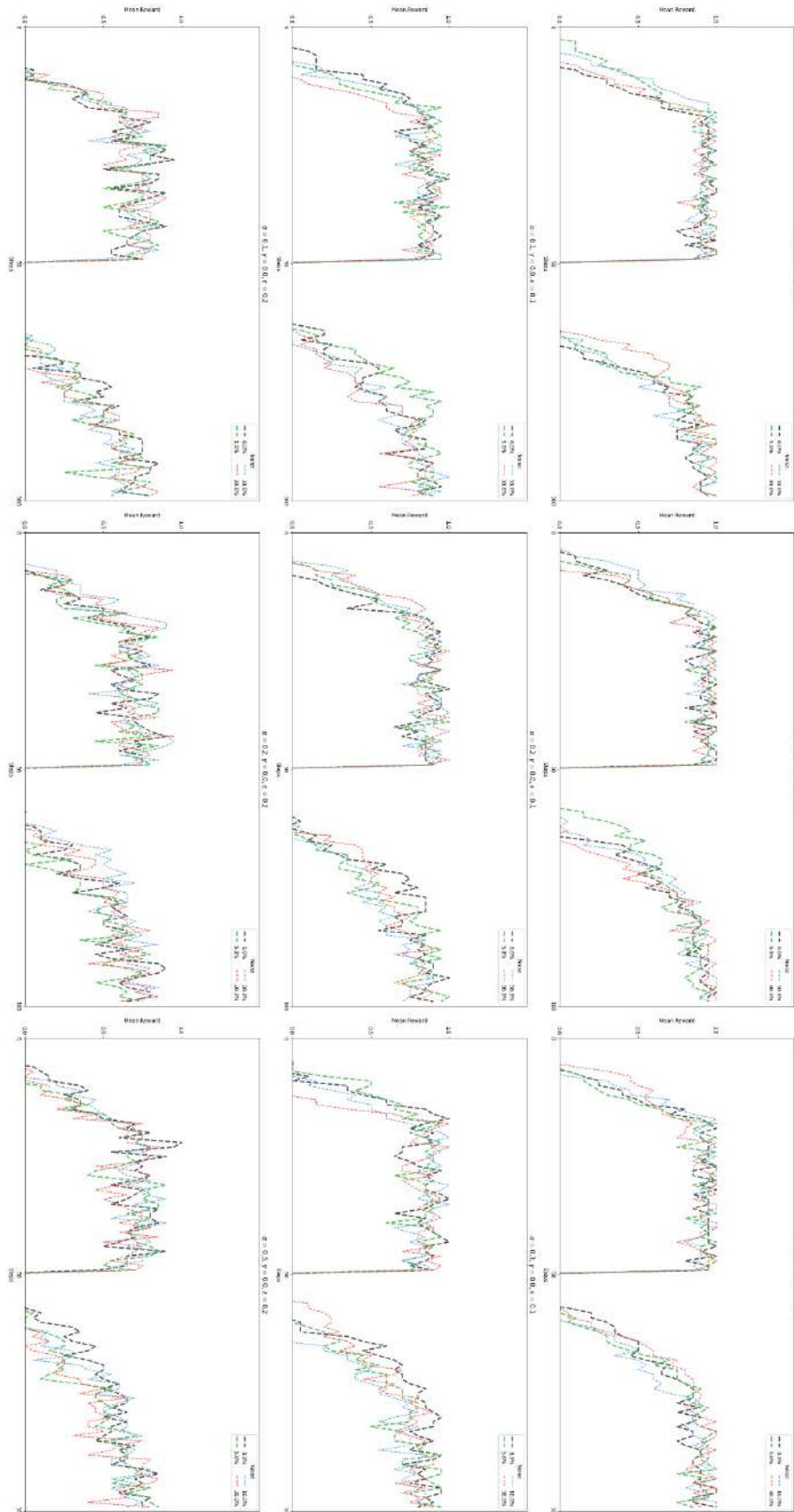
To be easily accessible to all partners and also to be as generic as possible, we implemented the finalized Reinforcement Learning as a network component. So for future behaviour modelling additional social signals can be added. Also the other components can independently be improved, which also will improve the reinforcement learning component.



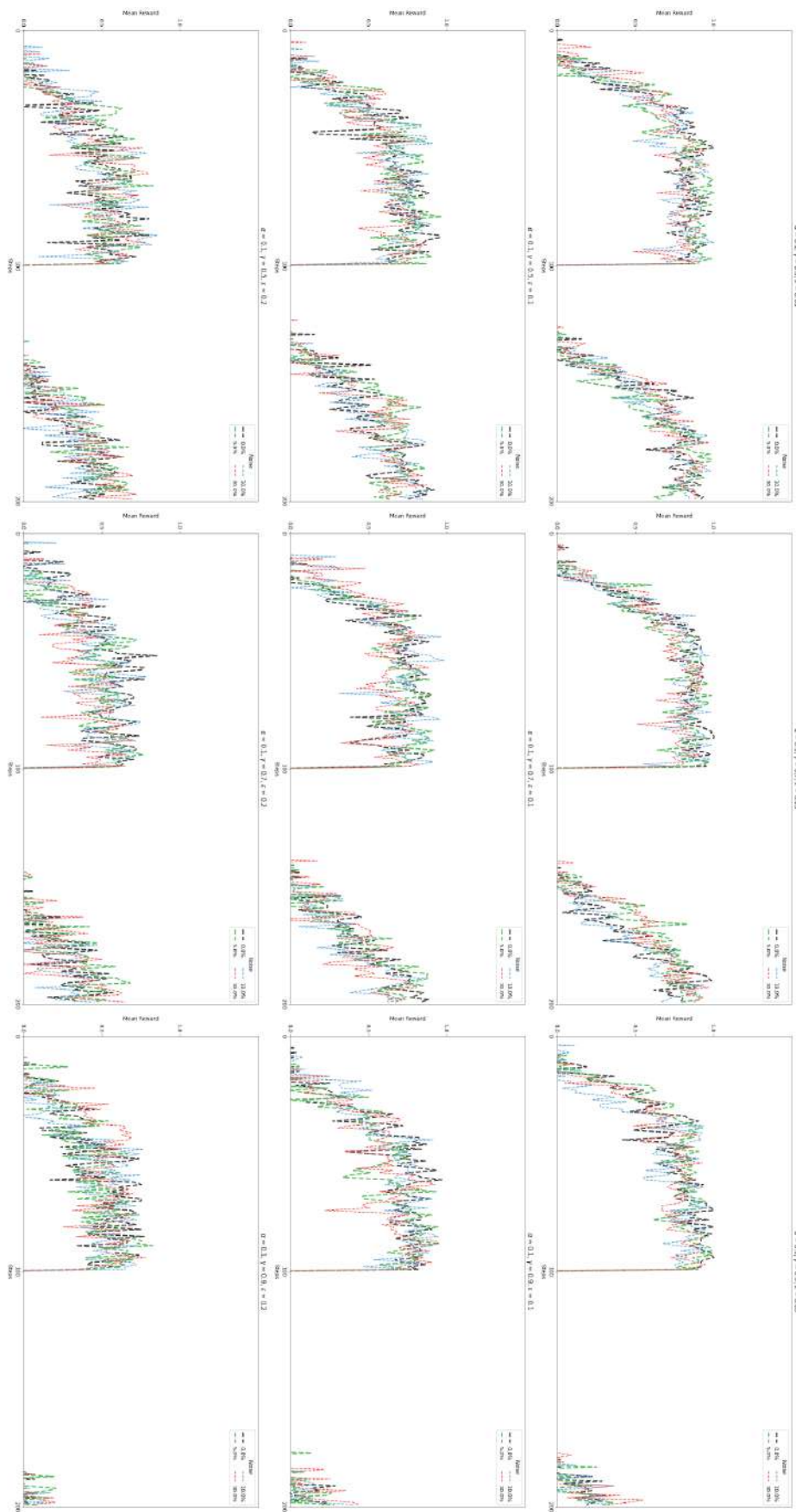
## 8 REFERENCES

- [1] R. S. Sutton und A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.
- [2] L. Busoniu, R. Babuska, B. De Schutter, und D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*, 0 Aufl. CRC Press, 2010.
- [3] D. Silver, „Lecture 6: Value Function Approximation“, S. 56.
- [4] G. Konidaris und S. Osentoski, „Value function approximation in reinforcement learning using the Fourier basis“, 2008.
- [5] G. Konidaris, S. Osentoski, und P. Thomas, „Value Function Approximation in Reinforcement Learning Using the Fourier Basis“, gehalten auf der Twenty-Fifth AAAI Conference on Artificial Intelligence, Aug. 2011. Zugegriffen: 11. Februar 2022. [Online]. Verfügbar unter: <https://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3569>
- [6] R. S. Sutton u. a., „Fast gradient-descent methods for temporal-difference learning with linear function approximation“, in *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, Montreal, Quebec, Canada, 2009, S. 1–8. doi: 10.1145/1553374.1553501.
- [7] L. Tickle-Degnen und R. Rosenthal, „The Nature of Rapport and Its Nonverbal Correlates“, *Psychol. Inq.*, Bd. 1, Nr. 4, S. 285–293, Okt. 1990, doi: 10.1207/s15327965pli0104\_1.

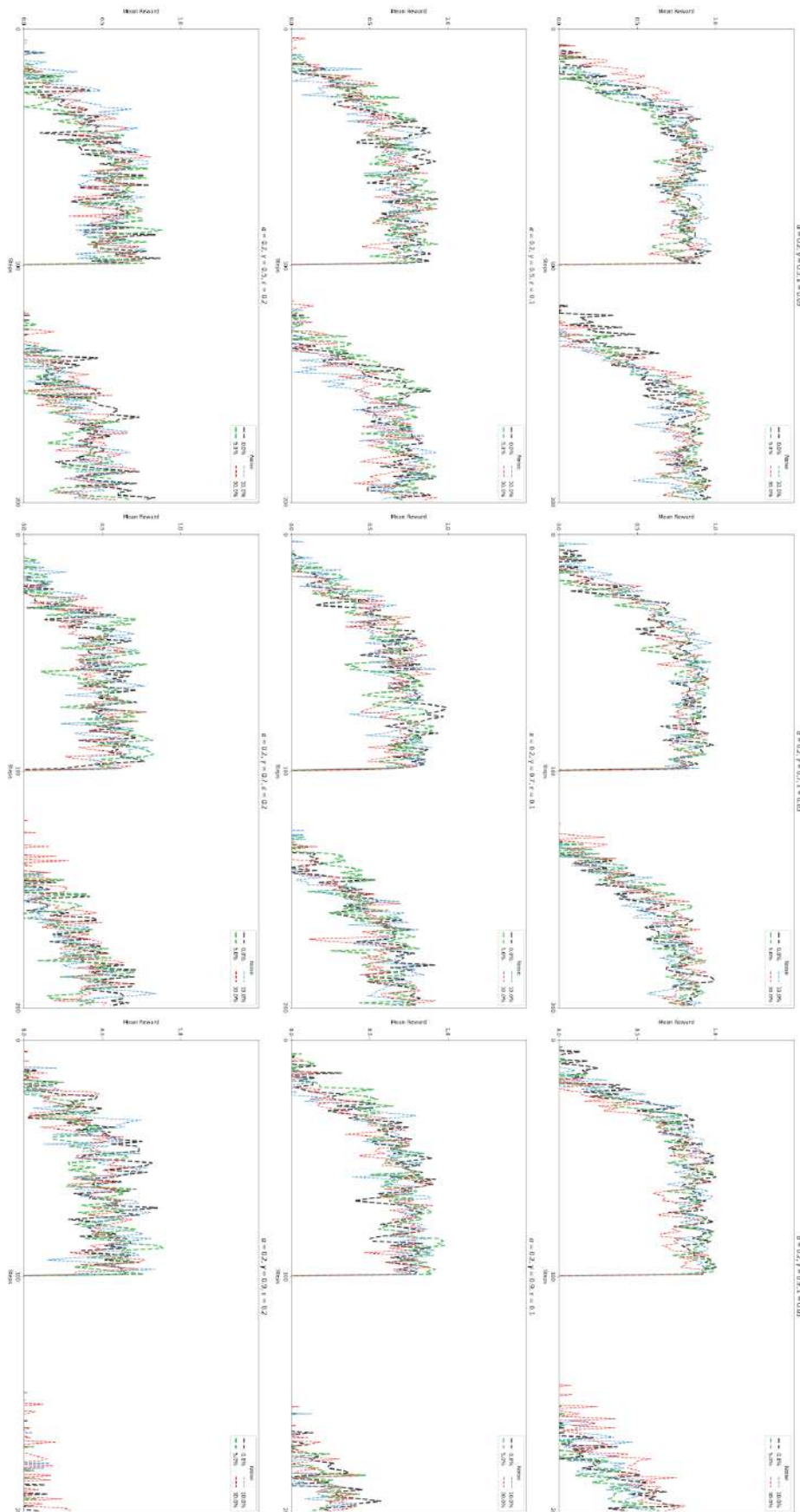
## 9 APPENDIX



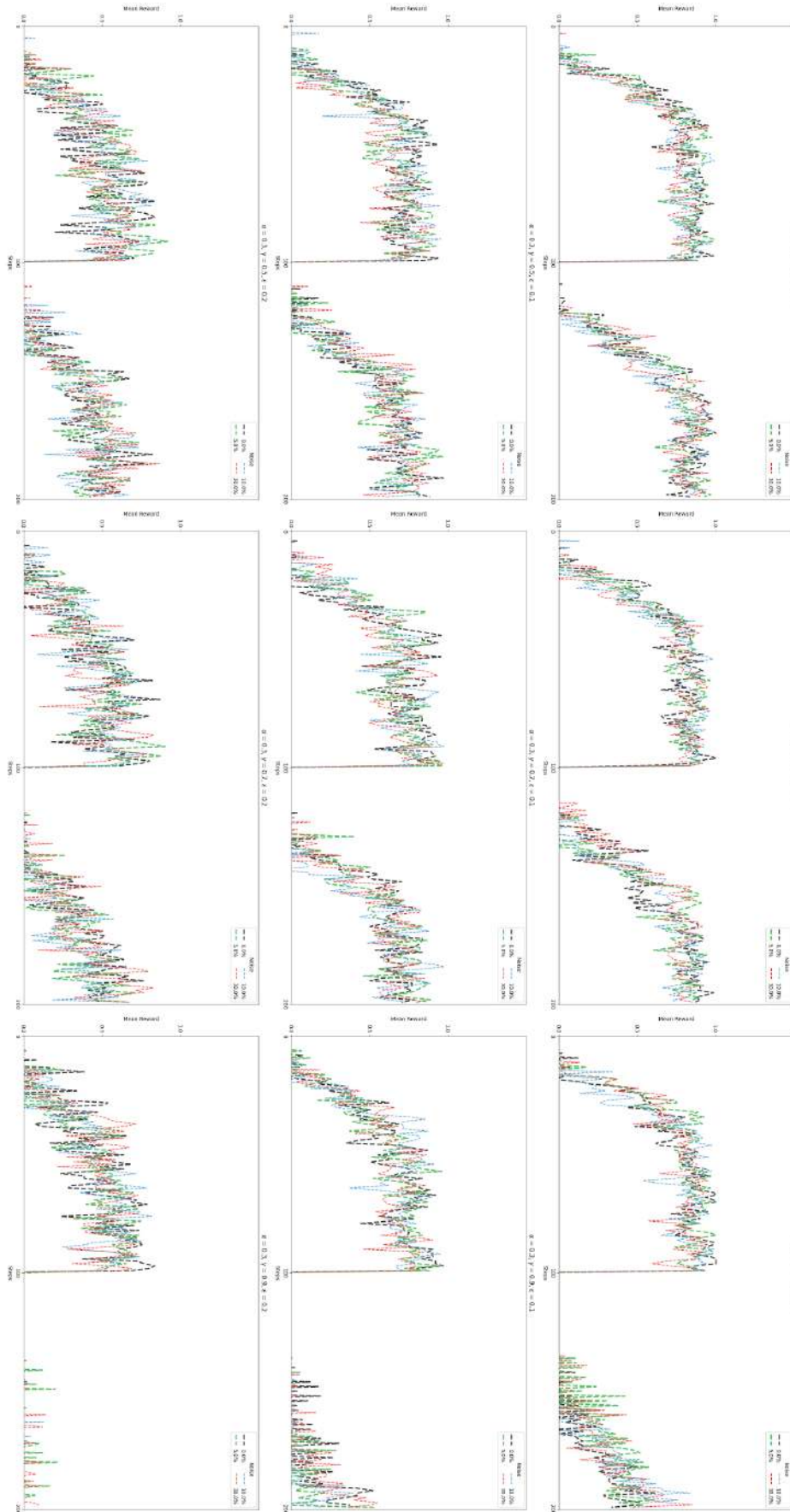
**Appendix figure 1: K-Armed Bandit experiments**



**Appendix figure 2: Q-Learning experiments ( $\alpha = 0.1$ )**

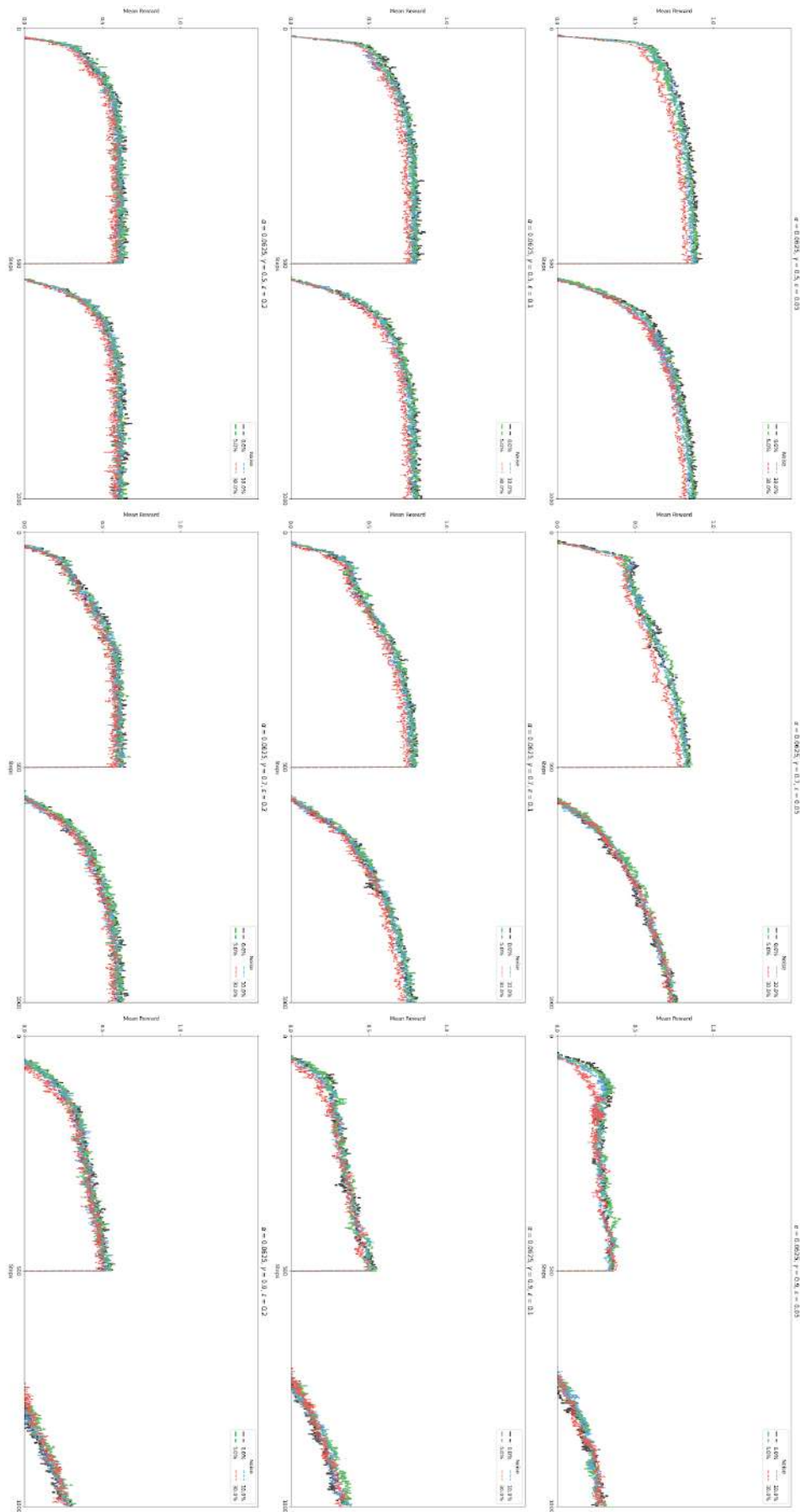


**Appendix figure 3: Q-Learning experiments ( $\alpha = 0.2$ )**

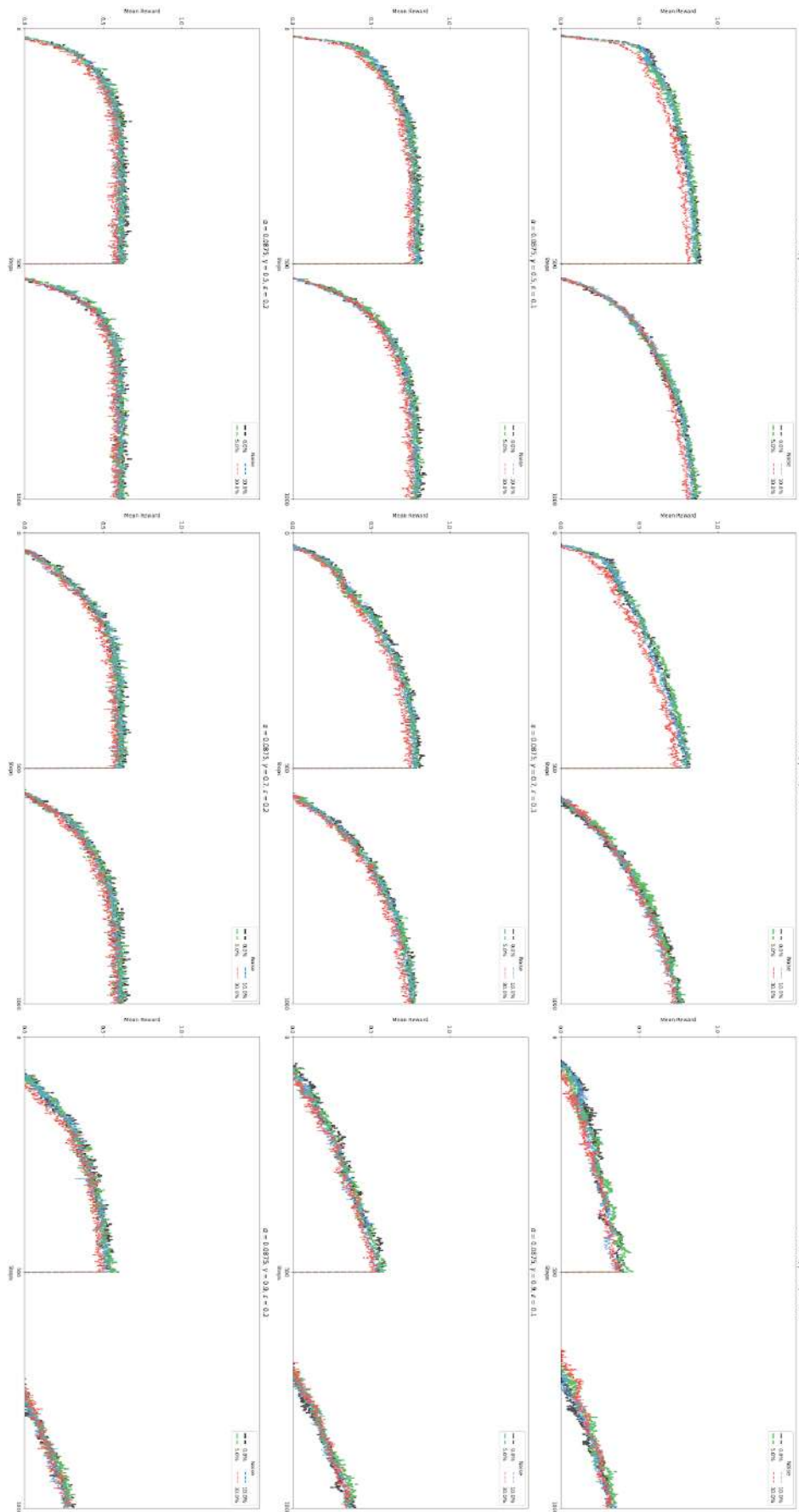


**Appendix figure 4:** Q-Learning experiments ( $\alpha = 0.3$ )



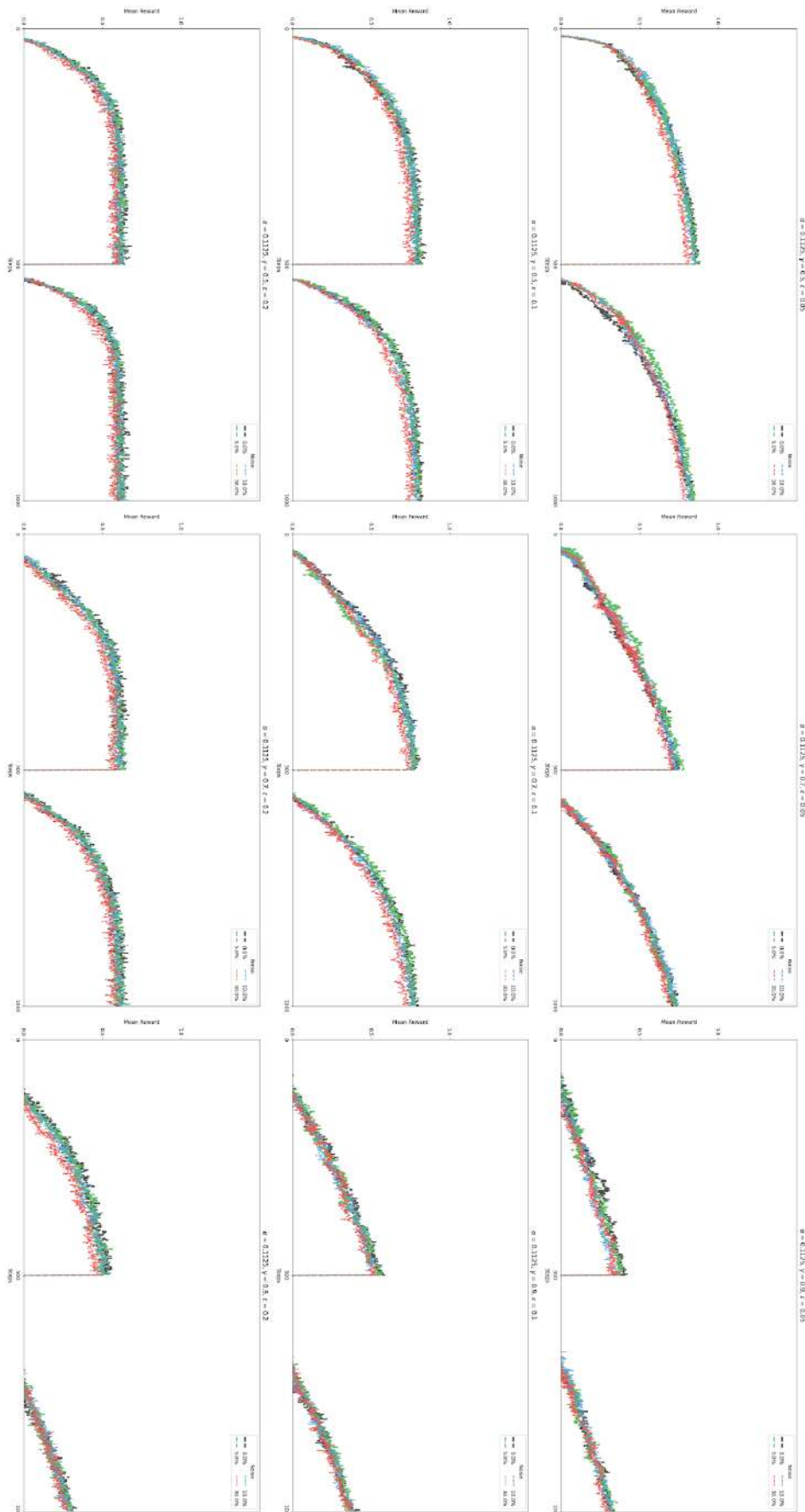


Appendix figure 5: TDC experiments ( $\alpha = 0.9\beta$ )



Appendix figure 6: TDC experiments ( $\alpha = 0.7\beta$ )





Appendix figure 7: TDC experiments ( $\alpha = 0.5\beta$ )