UNIVERSITY
OF TWENTE.

Designing an Adaptive Machine Learning Module for Robot Social Behavior Acquisition

Arjan Faber, Bob Schadenberg, Hideki Garcia Goo, Vanessa Evers and Gwenn Englebienne

Contents

1 Introduction

In recent years, the integration of robots into various aspects of society has accelerated, leading to increased interactions between humans and artificial agents. As robots transition from controlled environments to dynamic social settings, understanding and modeling social behavior becomes crucial for enhancing collaboration and fostering acceptance. In this context, reinforcement learning (RL) emerges as a promising avenue for facilitating the development of socially adept robotic systems.

In particular, Baraka and Veloso (2015) introduced three dynamic long-term user preference models, incorporating boredom, appreciation for change, and surprise, with an algorithm enabling parameter learning based on user feedback sequences. The paper Duchetto and Hanheide (2022) introduced a reinforcement learning approach for real-time behavioral adaptation of social robots during museum tours, using user engagement detected from the robot's camera as state and reward signals. Experimental results over a three-year period, including the COVID-19 pandemic, demonstrated that the adapted framework led to longer tours and decreased user disengagement compared to static policies. However, limitations were identified in handling high-dimensional state and action spaces, necessitating further research for more complex applications and inclusive interactions. In addition, Ritschel et al. (2017) found that by using reinforcement learning and social signals to adapt a social robot's personality to the preferences of human users, demonstrated through storytelling interactions based on "Alice in Wonderland" where the robot's extraversion is manipulated. The online learning process optimizes the robot's personality to maintain user engagement, leveraging multimodal social signal data as reward, and our future research aims to explore methods for effectively utilizing implicit and explicit user feedback to improve adaptation over prolonged interactions. Furthermore, Akalin and Loutfi (2021) discusses how RL approaches, commonly tested in virtual game environments with well-defined goals, can be applied in social robotics, where goals are less clear but social cues provide transparency into the robot's internal states. It suggests combining various reward approaches, such as intrinsically motivated methods and task performance-driven methods, to address the ambiguous reward problem and proposes integrating emotional models into robots' decision-making mechanisms for improved performance in real-world human-robot interaction scenarios. Vroon et al. (2019) studied detection and interpretation of social feedback cues related to approach distance. An extensive dataset was collected by manipulating approach distance and environment noise while measuring perceived appropriateness and tracking temporal positioning information. Surprisingly, neither approach distance nor environment noise significantly influenced perceived appropriateness, possibly due to participants compensating for robot behavior. The study provides a starting point for future research in similar areas, offering insights into dataset collection methods and potential cues.Finally, the paper Xu et al. (2023) explored the impact of various social cues on users' social presence and trust in social robots. While these cues, including appearance, voice, movement, and facial expressions, generally had small-sized effects individually, collectively they led to greater levels of social presence and trust when the Human-Robot Interaction (HRI) experience was designed to be more humanlike and intuitive.

In this paper, our research objectives are discussed in chapter 2, after which in chapter 3 the data collection is discussed. Then, in chapter 4 the methodologies are discussed. A conclusion is provided in chapter 5.

2 Research objectives

The main goal of our research is to develop the capacity for the robot to learn which multimodal behavior is appropriate in various situations. To achieve this, the robot must be able to generate combinations of different types of multimodal behavior, detect the social situation it is in, recognize when a response is needed, evaluate the reactions of the people it interacts with, and interpret the behavior as a reward or punishment.

The first challenge is to create a diverse range of multimodal behaviors that are natural and contextually appropriate. This is important because the robot must exhibit behavior that is coherent and sensitive to the context to be perceived as intelligent and socially competent. This project focuses on developing algorithms that allow the robot to combine various behavioral cues, such as gestures, speech, and facial expressions, in a fluid and context-sensitive manner.

Another challenge is accurately identifying the social context from limited and potentially ambiguous sensory input. Understanding the social situation is crucial for the robot to respond appropriately and engage effectively with humans. The project aims to enhance the robot's sensory processing capabilities and context-awareness, enabling it to better recognize and interpret social cues.

Determining the precise moments when a response is necessary and what type of response is appropriate is also challenging. Timely and appropriate responses are key to maintaining natural and effective interactions. This aspect will be addressed by developing decision-making frameworks that allow the robot to evaluate the need for a response based on the detected social context and interaction cues.

Accurately interpreting human reactions, which can vary widely and be subtle or non-verbal, is another challenge. Understanding human feedback is essential for the robot to learn and adapt its behavior effectively. The project includes developing machine learning models that can interpret a range of human reactions, from facial expressions to body pose, to refine the robot's behavior.

Finally, correctly interpreting feedback as positive (reward) or negative (punishment) to guide learning is critical. This interpretation is crucial for the robot's learning process, allowing it to adjust its behavior based on past interactions. The project focuses on reinforcement learning techniques that enable the robot to learn from interactions by evaluating feedback and adjusting behavior accordingly.

By addressing these challenges, the project aims to create a robot capable of engaging in natural, context-aware social interactions, significantly advancing the field of social robotics.

3 Data

3.1 Data Collection

Data collection involved using the Kuka iDo robot equipped with sensors (LiDAR) and cameras. Work regarding recovering a Kuka iDo and its wide range of capabilities can be found in the papers Mol (2023) and ten Berge (2022).

We stored raw data from Kinect and two realsense camera data (forward facing and downwards facing), both rgb and depth, from the recorded video streams. With the video data we used OpenPose to collect additional skeleton data after the sessions with the participant. With this our goal was to capture facial expressions, body pose, and distance to the robot. Also, with the combination of OpenPose and sensors we wanted to include a boolean feature measuring if the robot experiences a blockage of the way by people. We conducted our experiments in the hallway of the Design Lab at the UT campus where we wanted to collect data for three pre-defined events defined by the researchers that the robot can experience while the robot navigates through the hallway of a hospital. That is, 1. robot responding to blocking of the way of people by signaling help, 2. error mitigation by means of recovering from a sudden stop when a person navigates the robot from one point to another point, 3. robot adapting to people by acknowledge the person's presence and responding to a person attempt to interact with the robot. Our goal was to collect the data of the streams mentioned earlier capturing the natural responses of the participant interacting with the robot.

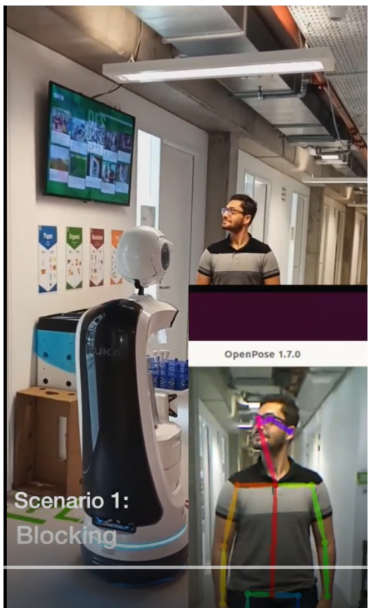


Figure 1: Event 1: Blocking the way

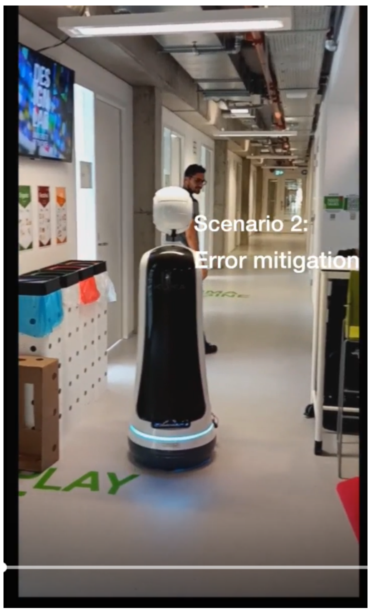


Figure 2: Event 2: Error mitigation

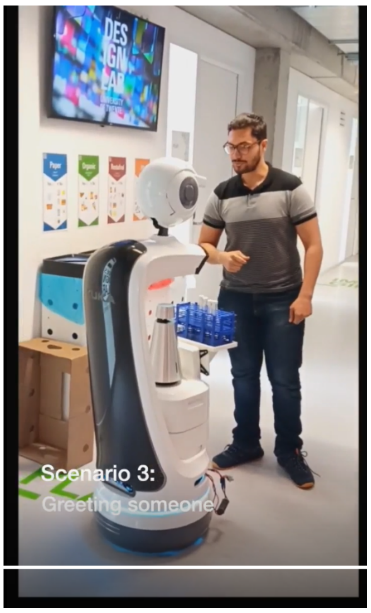
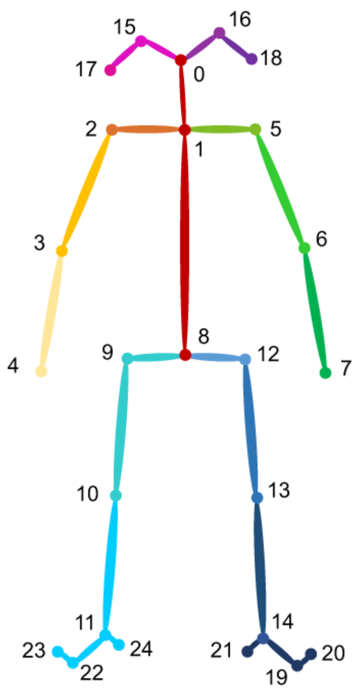


Figure 3: Event 3: Greeting someone

The complete data set that is collected is used for a learning from demonstration machine learning module to speed up the process of the self-learning process of the robot. We recruited participants who are staff members at UT and have limited familiarity with robots, in order to observe their natural responses to robot behaviors, mirroring the diversity of reactions we might expect from randomly selected individuals. Each session took 15 minutes total time. It includes an introduction with instructions, simulating the three scenarios and then debriefing how we will use the information gathered during the session. In total we collected data during 8 unique sessions implying 8 unique participants in total. Hence, 24 data sets (simulating our 3 events for each participant during each session). After data cleaning, we dropped data collected from one participant and 2 datasets of two events for another candidate, resulting in a total of 19 datasets with 7 unique participants in total. We did initially one recording for each event per participant.

However, in some cases we decided to re-do the recording of events. The features we used for determining the states and action pairs are:

- 'person_id' : Integer, referring to a unique person detected by OpenPose.
- 'body_part_id': Integer, referring to the body parts of person 'person_id' where body parts are mapped as follows:



0 - Nose	13 - Left knee
1 - Neck	14 - Left ankle
2 - Right shoulder	15 – Right eye
3 - Right elbow	16 – Left eye
4 - Right wrist	17 – Right ear
5 - Left shoulder	18 – Left ear
6 - Left elbow	19 - Left big-toe
7 - Left wrist	20 - Left small-toe
8 - Mid hip	21 - Left heel
9 - Right hip	22 - Right big-toe
10 - Right knee	23 - Right small-toe
11 - Right ankle	24 - Right heel
12 - Left hip	

Figure 4: OpenPose mapping (Mingming et al. (2023))

- 'pixel_x' : float, referring to the x-coordinate of the recognized 'body_part_id' of person 'person_id'.
- 'pixel_y': float, referring to the y-coordinate of the recognized 'body_part_id' of person 'person_id'.
- 'point_z': float, referring to the z-coordinate of the recognized 'body_part_id' of person 'person_id'.
- 'score': float, referring to a value between 0 and 1 indicating the confidence score of the keypoint detection.
- 'blocked_path': boolean, referring to the detection of people and/or objects blocking the path.

The label variable is 'sound'. That is, the sound that should be played on the Kuka iDo robot when the robot recognizes events occuring learned from the data set by LfD. In total there are 7 categories for the sound: 'happy','hello','no_sound', 'oops', 'oh_surprised_block', 'request_assistance' and 'thankyou-bye'. During data collection the researcher decided the appropriate sound the Kuka robot should choose during interaction with the participant in each of the 3 events.

3.2 Data Description

Let us now look at the final data set that we use to train our ML module. That is, we merge the 19 datasets of the 7 unique participants as mentioned in the previous section.

In table 1 it can be seen that we have 3323450 observations (state-action pairs). Table 2 summarizes the frequency of each category in the label variable during data collection.

Statistic	Value
count	3323450
unique	7
top	no_sound
freq	3053750

Table 1: General summary statistics label variable

sound	count
happy	23275
hello	85450
no_sound	3053750
oops	12000
oh_surprised_block	101900
request_assistance	16125
thankyou-bye	30950

Table 2: Frequency per category of the label variable

Similarly, we can take a look at summary statistics of relevant features in our data set. Table 3 gives an overview of the frequency of each category within relevant features in our data set. It is interesting to see that the robot not always interacted or acknowledged the presence of only one person but in fact it detected also multiple persons. That is, during most of our data collection sessions the robot interacted with one person ('person_id' equals 0) but also with multiple people (up to 5 'person_id' $\in [0, 4]$). In particular, there are 2736825 observations where OpenPose detects one person, of which 381775 observations a second person was detected, of which 161800 a third person was detected, of which 40325 a fourth person was detected and of which 2725 observations a fifth person was detected. Then, when you look at figure 5 it can be seen that the confidence score of the keypoints detected of a fourth and fifth person are spread out. Whereas, when you look at the confidence score of 1-3 persons the distribution of the confidence score matches a right-skewed distribution. It can be seen that when going from 1 up to 5 persons more outliers seem to appear. Hence, OpenPose was in general more confident in detecting keypoints for 1 person than for multiple persons as is expected. For more than 3 persons it seems that OpenPose does not perform well in general. When we look again at table 3 we can also look at the keypoints of the skeleton of detected persons. In total, it can be seen that OpenPose stores 132938 skeletons out of the total footage of frames that is collected from the Kinect camera. It is interesting to observe that out of the 2736825 observations that at least one person was detected OpenPose stored 132938 observations of skeleton data. Note that, OpenPose is computationally expensive and requires significant CPU and GPU. For data collection, we set the parameter to `-net_resolution -1x128` for allowing the laptop to run OpenPose on ROS Noetic.[1]. Further adjusting the parameters for OpenPose may lead to better and more accurate performance. As expected, there are no deviations between the frequencies of the body parts detected. That is, OpenPose detects complete skeletons (each part of the body obtains a (x, y, z) coordinate)).

[1]The used laptop is a Lenovo ThinkPad P16v Gen 1 with Intel Core i7-13700H and GPU Intel Iris Xe Graphics G7 96EUs

Feature	**Value**	**Count**
person_id	0	2736825
person_id	1	381775
person_id	2	161800
person_id	3	40325
person_id	4	2725
body_part_id	0	132938
body_part_id	1	132938
body_part_id	2	132938
body_part_id	3	132938
body_part_id	4	132938
body_part_id	5	132938
body_part_id	6	132938
body_part_id	7	132938
body_part_id	8	132938
body_part_id	9	132938
body_part_id	10	132938
body_part_id	11	132938
body_part_id	12	132938
body_part_id	13	132938
body_part_id	14	132938
body_part_id	15	132938
body_part_id	16	132938
body_part_id	17	132938
body_part_id	18	132938
body_part_id	19	132938
body_part_id	20	132938
body_part_id	21	132938
body_part_id	22	132938
body_part_id	23	132938
body_part_id	24	132938
blocked_path	Path is blocked. Obstacle detected.	1217900
blocked_path	Path is clear. Safe to move forward.	2105550

Table 3: Frequency per category of the features person_id,body_part_id and blocked_path.

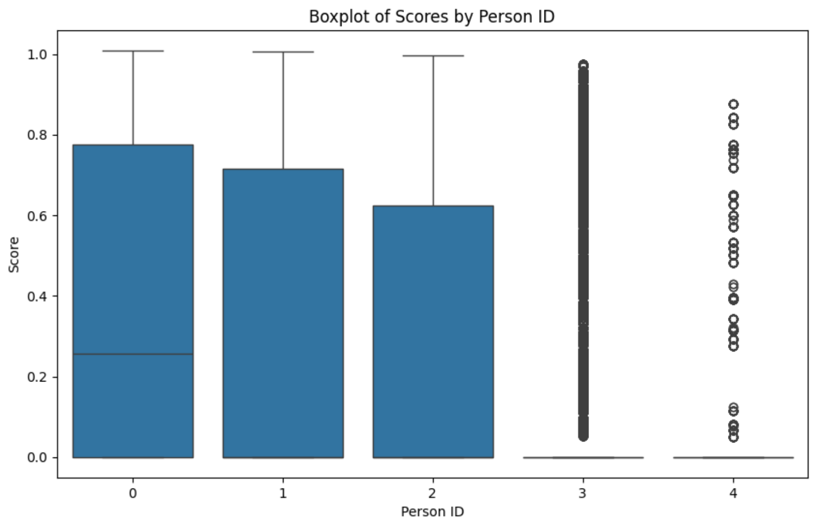


Figure 5: Distribution across person id

When we look at figure 6 it can be seen that the highest median are for the body-parts: nose (id: 0), right eye (id:15) and left eye (id:16). For the nose and the left eye but also the shoulders (id:1, id:2, id:5), the InterQuartile Ranges (IQR's) are also one of the lowest when comparing them with other IQR's of the body parts. Hence,the confidence scores seem to be more consistent for these body parts as well.

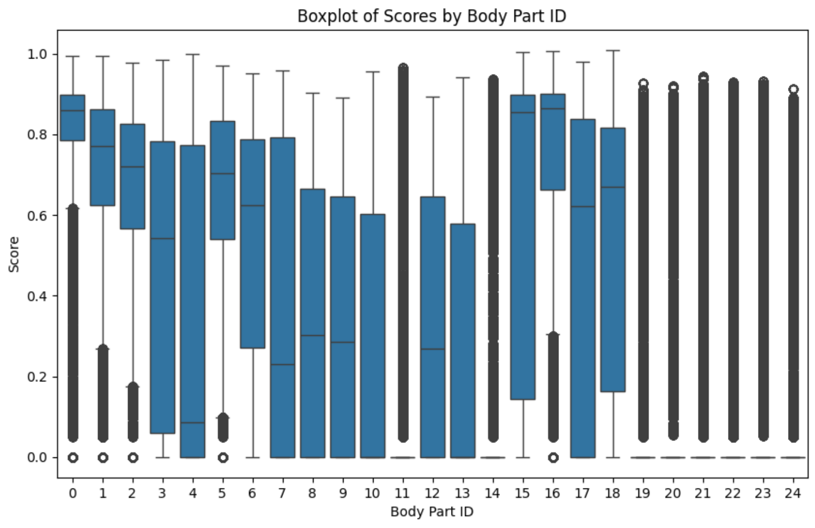


Figure 6: Distribution across body part id

Let us now consider being in a state where the robot plays a 'happy' sound. We can look at what happens in our data observations for this action when there is a blockage and not, and in particular the confidence scores of the body parts detected with OpenPose in this case. Firstly, let us consider a 'happy' action when there is no blockage by a person. It can be seen that for all the body parts there are still outliers but less than in general (that is, see figure 6). Also, it can be seen that the left-hand (id:7) has a significantly lower

IQR and overall lower confidence score than in the general case. Also, the left eye (id: 16) has a higher IQR that implies improved consistency in the confidence score. Overall, it can be seen that there are more body parts with a median confidence score below 0.6 than in the general case.

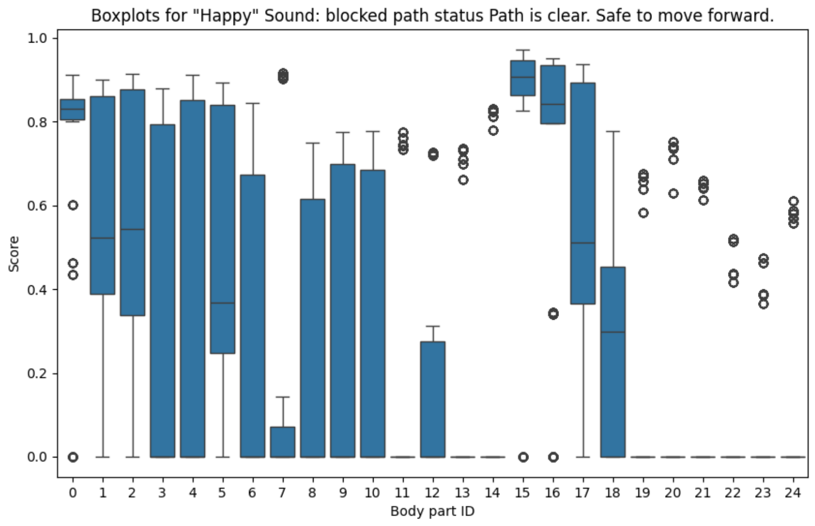


Figure 7: Distribution of category 'happy' for different body parts id and clear path.

Finally, let us consider the state where the robot plays a 'happy' sound when the path is blocked by a person. It can be seen in figure 8 that there are no outliers anymore. The nose, shoulder, elbow and right hand (id:0 - 5), eyes and ears (id:15-18) all have a confidence score above 0.5. Therefore, we can conclude that when a person blocks the path of a robot and the interaction is considered positive OpenPose consistently and confidently can detect the upper-part of the body-parts of a person.

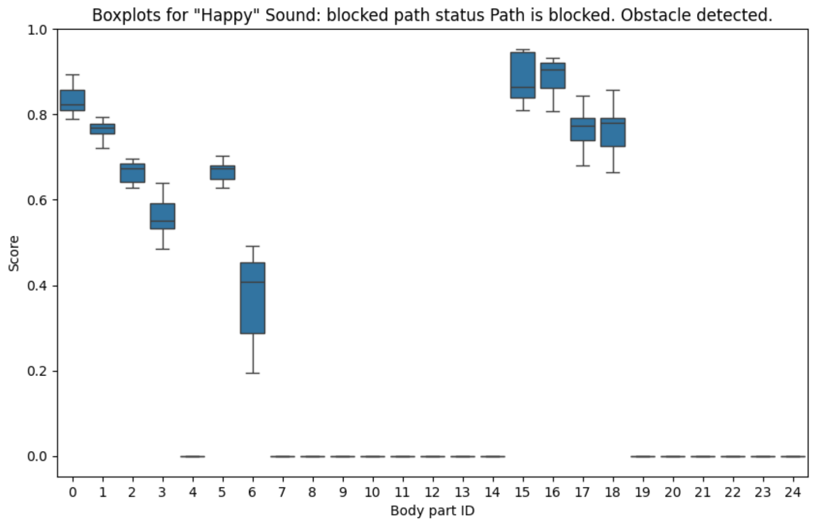


Figure 8: Distribution of category 'happy' for different body parts id and blocked path.

We can furthermore look at the mean confidence score of all the body parts of a person during each session over time. We find that 7 out of the total 28 recordings we did of the combined 7 different participants resulted in a time average confidence mean score above 0.5 as can be seen in figure 10. It is valid to conclude that lower body parts are

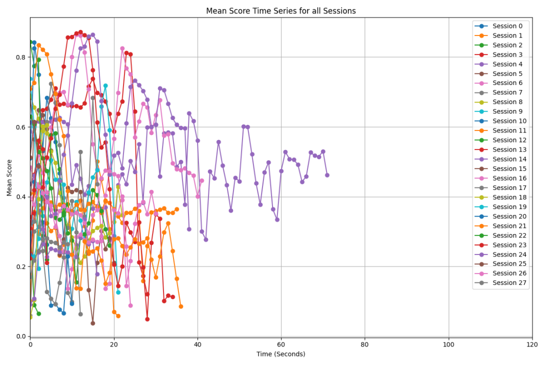


Figure 9: Mean confidence score for each session recorded

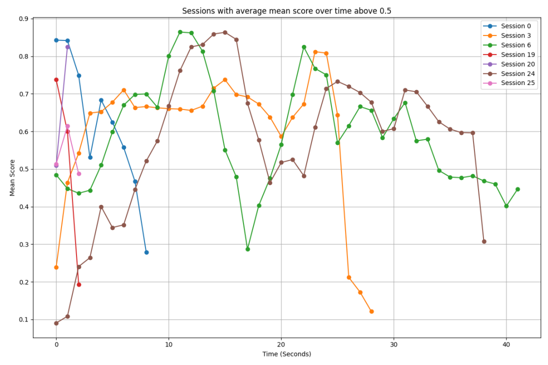


Figure 10: Mean confidence score of the sessions with average over time above 0.5

likely not being captured when the robot recognises a person standing close to the robot. In this case upper body parts are more likely to get higher confidence scores than lower body parts simply due to that the robot sees only the upper body part with the Kinect camera in the head of the robot. Hence, including lower body parts in calculating the mean confidence score results in lower output. Let us therefore also consider plotting the confidence scores over time during the sessions only considering the upper body parts (ID:0,1,2,3,5,15,16,17,18) and plotting only the results of the sessions where the time average mean confidence score is above 0.5. In figure 11 we can see that 20 sessions get a time-average mean confidence score above 0.5.

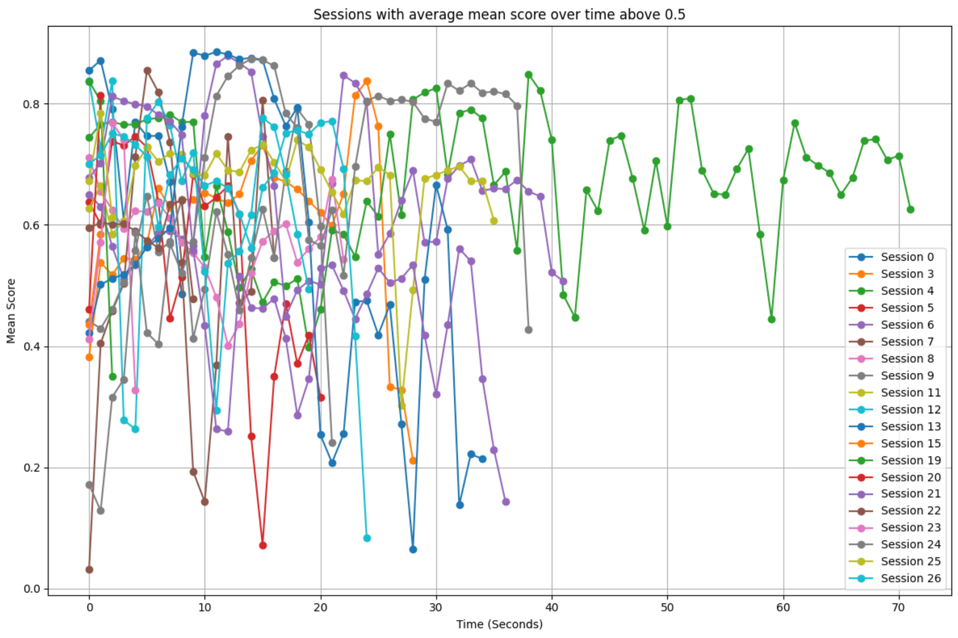


Figure 11: Mean confidence score of the sessions with average over time above 0.5 considering only upper body parts

4 Methodologies

4.1 Learning from Demonstration (LfD)

Learning from Demonstration (LfD) offers a powerful approach for transferring complex skills from humans to robots, enabling the rapid acquisition of expertise that might otherwise take years to master. Unlike reinforcement learning, where a policy is derived from trial and error experience, LfD involves deriving a policy from examples provided by a teacher.Formally, the world in LfD consists of states S and actions A. These states are observable and are mapped from S to Z by a mapping function M. A policy $\pi : Z \mapsto A$ is then formulated as a selection of actions based on the observable world states. One key consideration in LfD is whether to derive the policy after all training data is obtained (batch learning) or to incrementally develop it as data becomes available (interactive learning). The process of data acquisition in LfD can be categorized based on correspondence, as illustrated in Figure 12. $I(z, a)$ means identity function (direct mapping), while $g(z, a)$ is a mapping function used for correspondence.

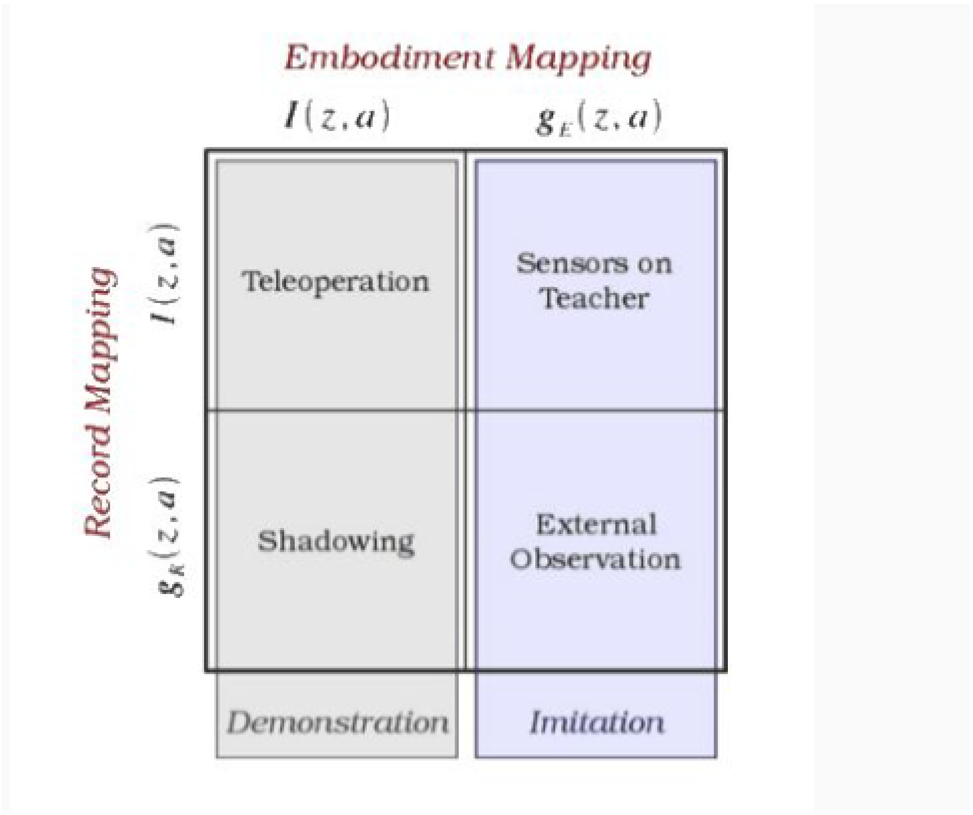


Figure 12: Data Acquisition: Correspondence Categories

In the context of LfD, teleoperation involves a human operator controlling a robot teacher, while shadowing entails a robotic platform mimicking the actions of a human teacher. Sensors placed directly on the teaching platform alleviate record correspondence issues in LfD. Furthermore, external observation involves using sensors external to the body executing the demonstration to record data. To derive a policy in LfD, a mapping function is essential. This function attempts to generalize over a set of training data by calculating the underlying relationship between states and actions. Two main categories of mapping functions are classification and regression. In classification, inputs are categorized into discrete classes, with algorithms like k-Nearest Neighbors and Bayesian networks used to perform the classification. Classification can be applied to various levels of task complexity, from low-level robot movement to high-level complex actions.In contrast, regression maps demonstration states to continuous action outputs. Function approximation is performed on demand, mapping current observations to actions at runtime.Actions in LfD are composed of pre-conditions (the state that must exist before an action) and post-conditions (the state immediately after the action). Additionally, non state-action information, such as intentions and annotations, can be provided by the teacher to the learner, supplementing the demonstration data. In our research we assume the reward is observable e.g. social signals like smiling. The signals are however ambiguous. That is, smiling can be positive or negative.

Challenges and considerations are:

1. **Sample Complexity**: Inferring the reward function from demonstrations may require a large amount of data, particularly in high-dimensional or complex environments. Efficient algorithms and techniques for reward function estimation are essential to mitigate sample complexity issues. Additionally, collecting high-quality demonstrations is critical for accurate reward inference.

2. **Generalization**: Inverse reinforcement learning (IRL) algorithms must generalize beyond the observed demonstrations to perform well in novel situations. Techniques for feature selection, reward shaping, and regularization play a critical role in ensuring that learned policies are robust and adaptable to unseen scenarios. Generalization is particularly challenging in environments with ambiguous or deceptive rewards.

3. **Assumptions**: IRL methods often make assumptions about the structure and form of the reward function, such as linearity or smoothness. Careful consideration of these assumptions and their implications on the learned policies is necessary to avoid overly restrictive modeling assumptions. Sensitivity analysis and robustness testing can help assess the impact of these assumptions on the performance of IRL algorithms.

4.2 Maximum Entropy Inverse Reinforcement Learning (MaxEnt IRL)

Inverse reinforcement learning (IRL) addresses the challenge of obtaining the reward function from an agent's behavior from empirical observations. Unlike traditional reinforcement learning, which assume access to the reward function, IRL learns it from demonstrated trajectories or behaviors (A. and S., 2000). MaxEnt IRL presents a principled approach aimed at identifying the reward function that best aligns with observed behavior, while concurrently maximizing entropy (B. et al., 2008).

4.2.1 Background

Inverse reinforcement learning: In IRL, the objective is to infer the reward function $R(s)$ given observed trajectories $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T)$, where s denotes states and a denotes actions.
MaxEnt IRL: MaxEnt IRL strives to identify the reward function $R(s)$ that maximizes entropy, subject to constraints imposed by observed trajectories (B. et al., 2008).

4.2.2 MaxEnt IRL Objective Function: Linear reward function

We define the feature matrix $X \in \mathbb{R}^{m \times n}$, where m is the number of samples and n is the number of features. We also define the observed labels $y \in \mathbb{R}^{m \times z}$, where $y_{i,j} \in \{0, 1\}$ and z is the number of categories in the label variable for $i, j = 1..m, 1...z$.

We aim to find the parameters $\theta \in \mathbb{R}^{n \times z}$ that maximize the log-likelihood of the observed data under the model. This leads to the following maximization problem:

$$\theta^* = \arg\max_{\theta} \mathcal{L}(\theta)$$

where the log-likelihood $\mathcal{L}(\theta)$ is defined as:

$$\mathcal{L}(\theta) = -\sum_{i=1}^{m} \log(P(a_i|s_i;\theta))]$$

with m is the total number of samples, $P(a_i|s_i;\theta)$ is the predicted probability of action a_i given state s_i and parameters θ with θ are the parameters of the model.

The probabilities $P(a_i|s_i;\theta)$ are calculated using the softmax function applied to the scores obtained by dot product of feature matrix X and parameter matrix θ:

$$P(a_i|s_i;\theta) = \frac{e^{X_i \cdot \theta}}{\sum_{j=1}^{z} e^{X_i \cdot \theta_j}} \tag{1}$$

Where:

- X_i represents the feature vector of sample i.
- θ_j represents the j^{th} column (action) of the parameter matrix θ.

The negative log likelihood is minimized in order to learn the parameters θ. The reward function $R(s)$ is assumed to be linear under this assumption and is computed using the dot product of features and parameters θ (Brown, 2011). Parameters θ are optimized using the L-BFGS-B method to maximize the MaxEnt IRL objective function. Rewards are computed for each state s.

4.2.3 MaxEnt IRL Objective Function: Predicting reward function with ANN

A linear reward function may not adequately capture the complexities of our state-action pairs. Therefore, let us consider introducing an artificial neural network (ANN) that is not linear by definition. Given is a *reward_model*, *expert_trajectories* and all possible states N. That is, for the reward model we introduce an ANN model that predicts rewards for states. Furthermore, we have the trajectories of expert demonstrations and all possible states.

The objective is to find the parameters θ of the reward function (represented by the ANN model) that maximize the log-likelihood of the observed expert trajectories under the Maximum Entropy Inverse reinforcement learning (MaxEnt IRL) framework.

The objective function is given by:

$$\begin{aligned}\underset{\theta}{\text{maximize}} \quad \mathcal{L}(\theta) &= -\sum_{s \in S} \left(\sum_{a} \pi(a|s;\theta) \log \frac{\pi(a|s;\theta)}{p(a|s)} \right) \\ &= -\sum_{i=1}^{M} \left(\sum_{j=1}^{N} \text{expert\_rewards}_{ij} - \log \left(\sum_{j=1}^{N} \exp(\text{all\_rewards}_{ij}) \right) \right) \\ &= -\sum_{i=1}^{M} \left(\sum_{j=1}^{N} \text{expert\_rewards}_{ij} - \log(Z_i) \right)\end{aligned}$$

where:

- M is the total number of expert trajectories.
- N is the number of states.
- $\text{expert\_rewards}_{ij}$ is the reward predicted by the ANN model for state j in expert trajectory i.
- all_rewards_{ij} is the reward predicted by the ANN model for state j in all states.
- $Z_i = \sum_{j=1}^{N} \exp(\text{all\_rewards}_{ij})$ is the partition function for the ith state.

The reward function is represented by an artificial neural network(ANN) with the following architecture:

Layer	**Type**	**Details**
1	Dense	{64 units, activation='relu', input_shape=(input_dim,)}
2	Dense	{28 units, activation='relu'}
3	Dense	{64 units, activation='relu'}
4	Dense	{128 units, activation='relu'}
5	Dense	{64 units, activation='relu'}
6	Dense	{7 units, activation='softmax'} (Output)

Table 4: Sequential ANN Model Architecture

The first layer has 64 neurons with ReLU activation function. It also specifies the input shape, which is determined by the input_dim parameter. The second layer has 28 neurons with ReLU activation function. The third layer has 64 neurons with ReLU activation function. The fourth layer has 128 neurons with ReLU activation function. The fifth layer has 64 neurons with ReLU activation function. The final layer has 7 neurons with softmax activation function, which outputs the probabilities of the input belonging to each of the 7 classes.

Let $T = 100$ represent the number of training epochs.

For each epoch $t \in [0, T-1]$:

Compute the MaxEnt IRL objective using the current reward model θ_t and expert trajectories $\{\tau^E\}$ and all states S writing mathematically as:

$$\mathcal{L}(\theta_t) = -\sum_{s \in S} \left(\sum_{a} \pi(a|s;\theta_t) \log \frac{\pi(a|s;\theta_t)}{p(a|s)} \right)$$

Compute the gradients of the log likelihood with respect to the trainable parameters of the reward model using automatic differentiation:

$$\nabla_{\theta_t} \mathcal{L}(\theta_t)$$

Update the parameters of the reward model θ_t using the computed gradients:

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} \ \mathcal{L}(\theta_t)$$

where η is the learning rate.

4.3 K-Fold Cross-Validation with $K = 5$

In order to learn more about how models perform on unseen data, the data set is split into a training set and a test set. A popular choice is following the 80%/20% splitting rule. However, we would like to use all the data that is available. Therefore, a K-Fold cross-validation procedure with $K = 5$ is used instead of splitting the data into 80% training data and 20% test data.

More specifically, for each epoch, by using the K-Fold cross-validation algorithm, the models are trained K times, each time using $K - 1$ folds as the training set and the remaining fold as a test set. This ensures that each instance in the dataset is used once as a test set and $K - 1$ times as a part of the training set.

In K-fold cross-validation, the dataset is divided into K folds, and the model is trained and evaluated K times. Each time, one of the K folds is used as the validation set, and the remaining $K - 1$ folds are used as the training set. Here's how the process is mathematically described. Let $\{F_1, F_2, \ldots, F_K\}$ represent the K folds of the dataset. Let F_k be the validation set. Let $T_k = \bigcup_{i \neq k} F_i$ be the training set formed by combining all the other $K - 1$ folds.The model is trained using the training set T_k. The predictions for the validation set F_k are generated. Let $R^{(k)}$ be the matrix of predicted rewards for the k-th fold, where $R_{ij}^{(k)}$ represents the predicted reward for the i-th validation sample and the j-th class. Let $Y^{(k)}$ be the matrix of true labels for the k-th fold (one-hot encoded), where $Y_{ij}^{(k)}$ is 1 if the i-th validation sample belongs to the j-th class and 0 otherwise. Let n_k be the number of samples in the k-th fold. Compute the accuracy for the k-th fold:

$$\varepsilon_k = \frac{1}{n_k} \sum_{i=1}^{n_k} \mathbf{1} \left(\arg\max_j R_{ij}^{(k)} = \arg\max_j Y_{ij}^{(k)} \right)$$

where $\mathbf{1}(\cdot)$ is the indicator function that equals 1 if the argument is true and 0 otherwise. Calculate the average accuracy across all K folds:

$$\bar{\varepsilon} = \frac{1}{K} \sum_{k=1}^{K} \varepsilon_k$$

Let us then plot average accuracies $\bar{\varepsilon}^{(t)}$ against the corresponding epochs $t \in [0, T - 1]$ to visualize the model's performance over the epochs $T = 100$.

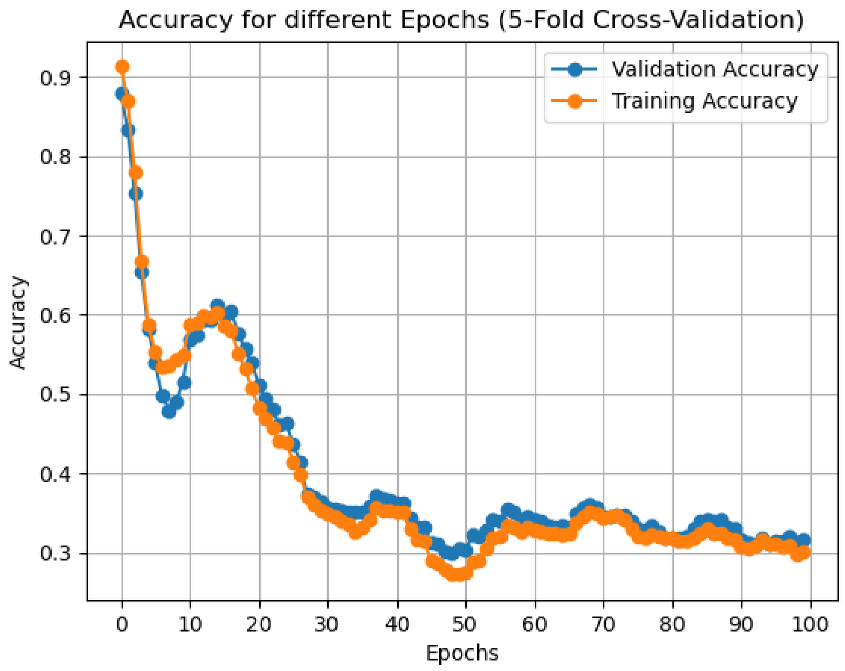


Figure 13: ANN predicted reward function: Accuracy for 100 epochs

The performance for the first 20 epochs seems good. However, since we have imbalanced data due to many data points labeled as 'no_sound' we need to consider other metrics. A more insightful metric is the macro-averaged precision. This metric measures the accuracy of positive predictions for class j in fold k. By calculating precision per class, we get insight into how well the model performs for each specific class. Each class is considered equally, highlighting if the model is performing poorly on less frequent classes. It ensures that the performance is not skewed by the distribution of classes. Mathematically, let us define:

- n_k: the number of samples in fold k
- C: the number of classes
- $Y_{ij}^{(k)}$: the true label for sample i in fold k for class j
- $P_{ij}^{(k)}$: the predicted label for sample i in fold k for class j
- $\mathbf{1}(\cdot)$: the indicator function

The precision for a single class j in fold k is

$$\delta_{j,k} = \frac{\sum_{i=1}^{n_k} \mathbf{1}\left(Y_{ij}^{(k)} = 1\right) \cdot \mathbf{1}\left(P_{ij}^{(k)} = 1\right)}{\sum_{i=1}^{n_k} \mathbf{1}\left(P_{ij}^{(k)} = 1\right)}$$

To compute the macro average precision for fold k, we average the precision values for all classes:

$$\hat{\delta}_k = \frac{1}{C} \sum_{j=1}^{C} \delta_{j,k}$$

Overall macro average precision across all K folds

$$\bar{\delta} = \frac{1}{K}\sum_{k=1}^{K}\hat{\delta}_k$$

Let us then plot average precision $\bar{\delta}^{(t)}$ against the corresponding epochs $t \in [0, T-1]$ to visualize the model's performance over the epochs.

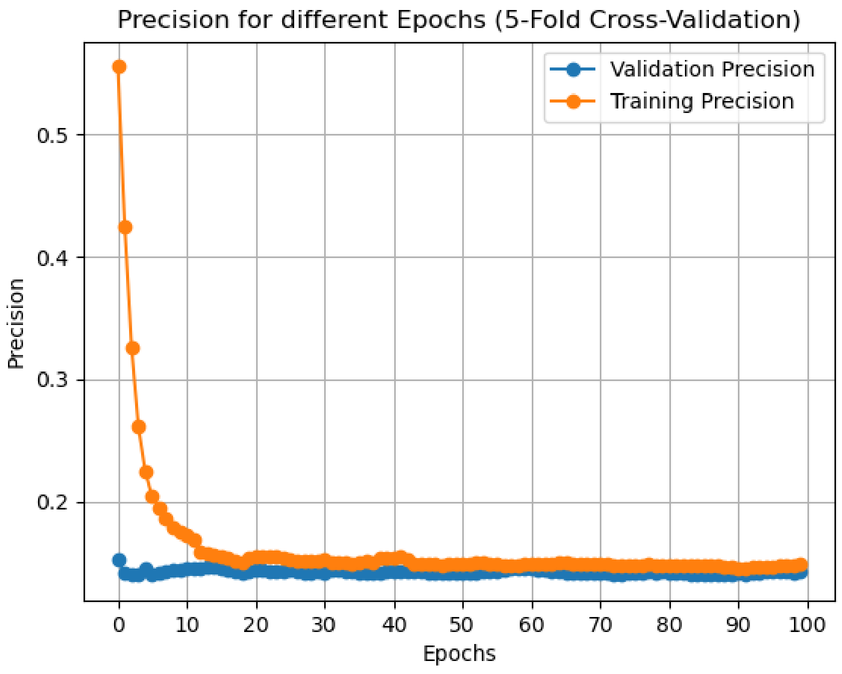


Figure 14: ANN predicted reward function: Macro averaged precision for 100 epochs

It can be seen that we obtain a low macro averaged precision score overall that improves slowly when we increase the epochs after 100 epochs. To improve performance we will consider a simplified ANN model where we only have 3 classes in the label variable. That is, we group the data points of the label variable `sound` into the categories: 'no_sound', 'positive_sound' and 'other'. Here, we store the sounds 'happy' and 'hello' in the 'positive_sound' and the rest of the sounds into the 'other' class. Also, we use the random oversampling technique to increase the number of data points that belong to the minority classes since approximately 92% of the data is labeled with 'no_sound' and the remaining 8% of the data observations with a sound.

Layer	Type	Details
1	Dense	{64 units, activation='relu', input_shape=(input_dim,)}
2	Dense	{28 units, activation='relu'}
3	Dense	{64 units, activation='relu'}
4	Dense	{128 units, activation='relu'}
5	Dense	{64 units, activation='relu'}
6	Dense	{3 units, activation='softmax'} (Output)

Table 5: Sequential ANN Model Architecture

Let us then plot average accuracies $\bar{\varepsilon}^{(t)}$ and average precision $\bar{\delta}^{(t)}$ against the corresponding epochs $t \in [0, T-1]$ to visualize the model's performance over the epochs $T = 100$.

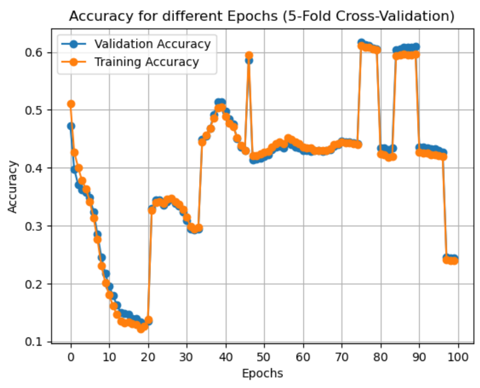


Figure 15: Accuracy

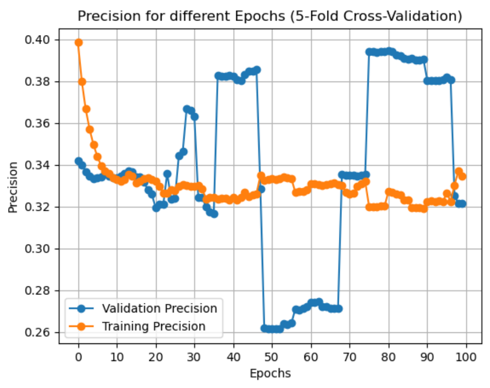


Figure 16: Macro averaged precision

We obtain a result that performs below a random model (probability $\frac{1}{3}$ of choosing a class). A possible argument why our models seem to perform poorly is that the timing when a sound is played is complex to predict. Therefore, we can decide to introduce a confidence interval such that before and after a particular sound is played within confidence interval the probability of this sound to be played should already be higher. Hence, we can model the probability by using a logistic regression model f that is trained to find the parameters $\mathbf{w}^*$ that best fit the data:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{i=1}^{m} \log\left(1 + \exp\left(-y_i(\mathbf{w}^\top \mathbf{x}_i)\right)\right)$$

We can then define the set of class labels $\mathcal{C} = \{\text{'positive'}, \text{'no\_sound'}, \text{'other'}\}$ and initialize the adjusted labels y_{adjusted} to be the same as the original labels y:

$$y_{\text{adjusted}} = y$$

Let $\mathbf{X} \in \mathbb{R}^{N \times n}$ be the feature matrix for the entire dataset, where N is the total number of samples. Compute the probability matrix P using the logistic regression model f:

$$P = f(\mathbf{X}) = [P_{ij}] \quad \text{where} \quad P_{ij} = P(y_i = j \mid \mathbf{x}_i)$$

Calculate the mean (μ) and standard deviation (σ) of the predicted probabilities for each class:

$$\mu_k = \frac{1}{N} \sum_{i=1}^{N} P_{i,k}$$

$$\sigma_k = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (P_{i,k} - \mu_k)^2}$$

where $P_{i,k}$ is the probability of class k for sample i.

Define the lower and upper thresholds for the confidence interval as $\mu_k - 2\sigma_k$ and $\mu_k + 2\sigma_k$, respectively.

For each sample i in the dataset, we then adjust the label based on the predicted probability and the confidence interval:

$$y_{\text{adjusted}} = \begin{cases} \text{'positive'} & \text{if } \mu_0 - 2\sigma_0 \leq P_{i,0} \leq \mu_0 + 2\sigma_0 \\ \text{'other'} & \text{if } \mu_1 - 2\sigma_1 \leq P_{i,1} \leq \mu_1 + 2\sigma_1 \\ \text{'no\_sound'} & \text{otherwise} \end{cases}$$

For each sample i, we store the index i and the probabilities p_{positive} and p_{other}. We update the labels y to be the adjusted labels y_{adjusted}:

$$y = y_{\text{adjusted}}$$

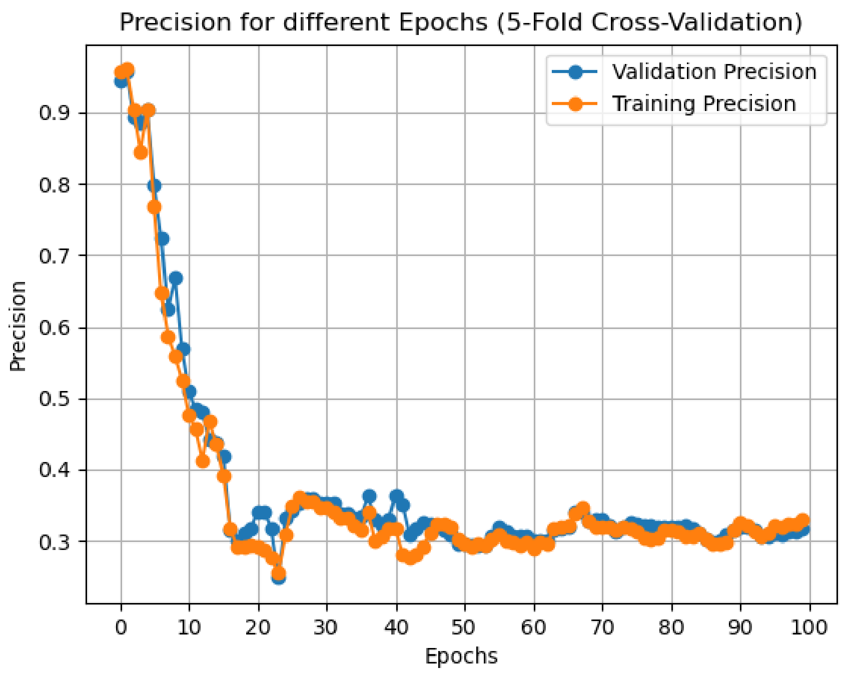


Figure 17: ANN predicted reward function: Macro averaged precision for 100 epochs adjusted labels

As we compare the initial performance of the simplified model with and without the adjusted labels(see figure 17 and 16) we can conclude that within 15 epochs we see a significant improvement for the case where we adjusted the label based on confidence intervals. That is, within the 15 epochs our model is able to perform better than random as we get a precision score above 0.4. The model seems to have converged, meaning it has learned the patterns in the data sufficiently well within the first 15 epochs. Further training does not provide additional benefit because the model has essentially captured all the learnable information from the data. We could have applied early stopping here.

Note that, our model still require improvements when we increase the training epochs but it does already provide a foundation to further develop a learning module for our robot.

4.4 Reinforcement Learning Environment and DQN Agent

We now use our obtained model for the reward function and present a mathematical model of the reinforcement learning environment (`CustomEnv`) and the Deep Q-Network Agent (`DQNAgent`). These components interact to learn an optimal policy using the Deep Q-Learning algorithm. The reinforcement learning environment is represented by the `CustomEnv` class, which interacts with a real-time data node and a pre-trained reward prediction model. Here, we describe its components mathematically. Let $\mathcal{S}$ denote the state space of the environment. Initially, the environment state $s \in \mathcal{S}$ is set to the merged dataset fitted by the real-time data node. The action space $\mathcal{A}$ consists of a set of actions available to the agent. In this case, $\mathcal{A}$ is represented as a 3-dimensional continuous space, $\mathcal{A} = [0, 1]^3$. The environment uses our earlier described pre-trained reward prediction model R to compute rewards based on the current state s:

$$r = R(s)$$

The environment evolves over time through the `step` function, which takes an action $a \in \mathcal{A}$ and returns the next state s', reward r, and a termination signal done:

$$(s', r, \text{done}) = \text{step}(s, a)$$

The step function enforces a minimum time interval between actions and skips action execution if the same action is repeated within a short period. The Deep Q-Network Agent utilizes a neural network to learn the optimal policy based on experiences collected from interacting with the environment. The Q-function $Q(s, a; \theta)$ estimates the expected cumulative reward from state s by taking action a, parameterized by weights θ. The agent learns Q through neural network training. To stabilize learning, the agent stores experiences $(s, a, r, s', \text{done})$ in a replay memory $\mathcal{D}$. The memory is sampled to update the Q-network parameters θ using the Bellman equation:

$$Q(s, a; \theta) \leftarrow r + \gamma \max_{a'} Q(s', a'; \theta)$$

where γ is the discount factor. The agent selects actions based on an epsilon-greedy policy:

$$a = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg\max_a Q(s, a; \theta) & \text{otherwise} \end{cases}$$

During training, the agent interacts with the environment, collects experiences, and updates the Q-network using batches of experiences sampled from $\mathcal{D}$.

5 Conclusion

The main goal of our research was to develop the capacity for the robot to learn which multimodal behavior is appropriate in various situations. We achieved this by developing algorithms that allow the robot to combine various behavioral cues, such as gestures, speech, and facial expressions, in a fluid and context-sensitive manner. The ANN model developed in this study represents a step towards enabling the robot to generate and recognize these multimodal behaviors. Another objective was to enhance the robot's ability to understand social context from limited and potentially ambiguous sensory input. Our work focused on improving the robot's sensory processing capabilities and context-awareness, which are essential for recognizing and interpreting social cues effectively.We aimed to develop decision-making frameworks that enable the robot to determine when and how to respond based on the detected social context and interaction cues. This objective was addressed by integrating reinforcement learning techniques, specifically Deep Q-Learning, to optimize the robot's policy in response to social interactions.Furthermore, our research aimed to develop machine learning models capable of interpreting a range of human reactions, from facial expressions to body pose, to refine the robot's behavior. The ANN model trained in this study was extended to handle diverse reactions and refine its responses accordingly. Lastly, we focused on implementing reinforcement learning techniques that allow the robot to interpret feedback as positive (reward) or negative (punishment), guiding its learning process. The development of the `DQNAgent` interacting with the `CustomEnv` class exemplifies our approach to leveraging reinforcement learning for this purpose.

In this study, we explored the development of a predictive model to classify sound events in a robotics application using an artificial neural network (ANN) and reinforcement learning techniques. The objective was to improve the model's performance on an imbalanced dataset where the majority class was 'no_sound'. We found that the dataset was highly imbalanced, with approximately 92% of the data labeled as 'no_sound' and only 8% as various sound classes. We employed random oversampling to address this imbalance, ensuring that the model had sufficient data to learn from the minority classes. As part of our inverse reinforcement learning model to obtain a reward function, we implemented a sequential ANN model with six dense layers, including ReLU activation functions for hidden layers and a softmax activation function for the output layer. A 5-Fold Cross-Validation approach was used to use the entire dataset for both training and validation purposes. Accuracy and macro-averaged precision were chosen as performance metrics. Accuracy measured the overall correctness of the model, while macro-averaged precision evaluated the model's performance across different classes equally, mitigating the effects of class imbalance. As for the model performance, the initial model demonstrated acceptable performance within the first 20 epochs, but the overall macro-averaged precision remained low due to the imbalanced dataset. We simplified the label categories to three classes: 'no_sound', 'positive_sound', and 'other'. This simplification, combined with random oversampling, improved the model's performance. The model, however, still performed below the baseline of random guessing initially. To further improve the model, we introduced confidence intervals around predicted probabilities, adjusting labels accordingly. This adjustment led to a significant improvement in macro-averaged precision within 15 epochs, indicating that the model was able to learn and generalize better with the refined labeling approach. We leveraged our predictive model to develop a reinforcement learning framework. The `CustomEnv` class and `DQNAgent` interact to learn an optimal policy using the Deep Q-Learning algorithm. Our research demonstrates that by addressing class imbalance through techniques like label simplification, random oversampling, and confidence interval adjustments, significant improvements in model performance can be achieved. The ANN model showed promising results, especially after label adjustments, performing better than random guessing within a short number of epochs. However, there is room for further enhancements. Future work can focus on implementing early stopping to prevent overfitting, exploring other advanced techniques for handling imbalanced data, such as SMOTE (Synthetic Minority Over-sampling Technique) and refining the reinforcement learning approach to optimize the agent's policy and further enhance model performance. Additionally, integrating uncertainty estimation methods like Bayesian neural networks could provide more reliable predictions in uncertain sound environments. Overall, this study provides a first step for developing robust sound event classification models in robotics applications, with potential extensions into real-time data integration and continuous learning environments.

6 References

A., N. and S., Y. . R. (2000). Apprenticeship learning via inverse reinforcement learning.

Akalin, N. and Loutfi, A. (2021). Reinforcement learning approaches in social robotics.

B., Z., Maas, D., and Bagnell, A. (2008). Maximum entropy inverse reinforcement learning.

Baraka, K. and Veloso, M. (2015). Adaptive interaction of persistent robots to user temporal preferences. volume 9388 LNCS.

Brown (2011). Inverse reinforcement learning with locally consistent reward functions.

Duchetto, F. D. and Hanheide, M. (2022). Learning on the job: Long-term behavioural adaptation in human-robot interactions. *IEEE Robotics and Automation Letters*, 7.

Mingming, Z., Yanan, Z., Xinye, X., and Ziwei, R. (2023). Multi-view emotional expressions dataset using 2d pose estimation. *Scientific Data*.

Mol, G. (2023). Developing a platform for the kuka ido social robot. *Creative Technology BSc thesis, University of Twente.*

Ritschel, H., Baur, T., and Andre, E. (2017). Adapting a robot's linguistic style based on socially-aware reinforcement learning. volume 2017-January.

ten Berge, C. (2022). Reverse engineer discontinued kuka-ido robots and make them applicable for educative use. *Robotics and Mechatronics BSc thesis, University of Twente.*

Vroon, J., Englebienne, G., and Evers, V. (2019). Detecting perceived appropriateness of a robot's social positioning behavior from non-verbal cues. *31st IEEE International Conference on Robot and Human Interactive Communication (RO-MAN).*

Xu, K., Chen, M., and You, L. (2023). The hitchhiker's guide to a credible and socially present robot: Two meta-analyses of the power of social cues in human–robot interaction. *International Journal of Social Robotics*, 15.