Applied Intelligence (2023) 53:27249–27268
https://doi.org/10.1007/s10489-023-04691-5

Variational meta reinforcement learning for social robotics

Anand Ballou[1] · **Xavier Alameda-Pineda**[1] · **Chris Reinke**[1]

Accepted: 4 May 2023 / Published online: 6 September 2023

Abstract
With the increasing presence of robots in our everyday environments, improving their social skills is of utmost importance. Nonetheless, social robotics still faces many challenges. One bottleneck is that robotic behaviors often need to be adapted, as social norms depend strongly on the environment. For example, a robot should navigate more carefully around patients in a hospital than around workers in an office. In this work, we investigate meta-reinforcement learning (meta-RL) as a potential solution. Here, robot behaviors are learned via reinforcement learning, where a reward function needs to be chosen so that the robot learns an appropriate behavior for a given environment. We propose to use a variational meta-RL procedure that quickly adapts the robots' behavior to new reward functions. As a result, in a new environment different reward functions can be quickly evaluated and an appropriate one selected. The procedure learns a vectorized representation for reward functions and a meta-policy that can be conditioned to such a representation. Given observations from a new reward function, the procedure identifies its representation and conditions the meta-policy to it. While investigating the procedure's capabilities, we realized that it suffers from posterior collapse where only a subset of the dimensions in the representation encode useful information, resulting in reduced performance. Our second contribution, a radial basis function (RBF) layer, partially mitigates this negative effect. The RBF layer lifts the representation to a higher dimensional space, which is more easily exploitable for the meta-policy. We demonstrate the interest of the RBF layer and the usage of meta-RL for social robotics in four robotic simulation tasks.

Keywords Social robotics · Meta-learning · Deep reinforcement learning · Radial-basis-functions

Since the emergence of social robotics [1], substantial efforts have been made to enable the deployment of interactive robots in different environments, such as industrial [2], healthcare [3], or education [4] facilities. However, this deployment is very challenging, since socially appropriate behaviors are strongly context-dependent, making it necessary to adapt them to each target environment. For example, a social robot might be used to navigate a care center for the elderly or an office. In the care center, the robot should avoid getting too close to people to ensure that they do not feel unsafe. Conversely, in an office environment this restriction might be less important and the robot may move closer to people to reach its target position faster. Being able to adapt a robot's behavior quickly to the specific needs of a new environment is essential to the practical application of social robotics.

Given the complex dynamics of social environments, the approach of manually designing and adapting behaviors for each environment, for example by finite state machines [5], is often not feasible. As an alternative, reinforcement learning (RL) [6] can be used for training social behaviors. However, classical RL introduces its own set of problems when applied to social robotics [7]. The most prominent problem is that RL requires large amounts of training data, meaning hours of observations that have to be collected while the robot is learning behaviors for a new environment.

We propose the use of meta-RL [8] to overcome this issue (Fig. 1). Meta-RL aims to adapt the learning process to a certain problem domain so that it can efficiently solve a new target task in this domain. Of particular interest to us are

✉ Anand Ballou
anand.ballou@inria.fr

Xavier Alameda-Pineda
xavier.alameda-pineda@inria.fr

Chris Reinke
chris.reinke@inria.fr

[1] RobotLearn, Inria, Univ. Grenoble Alpes, CNRS, Grenoble INP, LJK, 655 Avenue de l'Europe, Montbonnot 38334, France

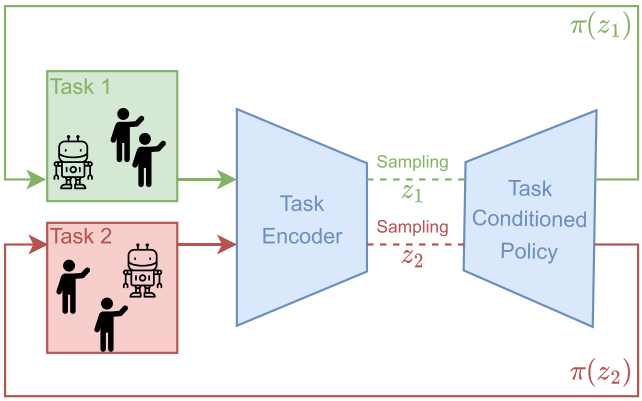


Fig. 1 Variational Meta-RL for social robotics: Meta-RL enables robots to quickly adapt to task changes using only a few observations of the target task, making it an efficient approach for training social behaviors in robots. We use a variational encoder network to convert task observations into a task representation in the form of a vector. The meta-policy, which is the robot's behavior, is then conditioned on this task representation. The robot can quickly adapt to a new task by collecting a small number of observations and computing the corresponding task representation. This representation is then used to condition the meta-policy for the new task

methods that are able to adapt using only a few observations of the target task. In our case, tasks represent the different environments of a social robot. Each task is defined by a reward function that defines which states a robot should try to reach or avoid. It is often a weighted sum over several reward components: $R(s) = \sum_i w_i r_i(s)$. For example, for social navigation [9, 10], the reward function might include a positive component for reaching the robot's target position and a negative component for getting too close to people. Depending on the environment, such as in our office vs. care center example, we choose a different reward function for each environment. In a care center, the reward function might have a stronger negative component to prevent the robot from getting too close to patients. Conversely, in an office environment, the positive component for reaching the goal might be stronger so that the robot reaches its destination faster. Meta-RL aims to adapt quickly to such task changes.

Even though meta-RL should efficiently adapt to new tasks (i.e. reward functions), to our knowledge its use in social robotics has scarcely been explored. To date, the only study on the topic uses on-policy meta-RL, requiring lots of interactions with the environment before reaching a satisfactory performance [11]. To cope with this issue, we propose to explore the use of off-policy meta-RL methods for social robotics tasks.

More precisely, we will focus on meta-RL methods that learn task-conditioned policies, since they are able to adapt using only a few observations (or environment steps). These methods learn a meta-policy, i.e., a behavior that is conditioned to a task representation in the form of a vector $z \in \mathbb{R}^d$. An encoder network that takes input observations of a task is used to compute z. The encoder network and the meta-policy are learned concurrently in an end-to-end way on a set of random source tasks. After this meta-training, given a new target task, a small number of observations are collected to compute its task representation and to condition the meta-policy to it.

A promising research direction for task-conditioned policies is variational architectures [12], such as PEARL [13]. Instead of learning a deterministic representation of tasks, PEARL learns to represent them via a distribution, exploiting the flexibility of probabilistic models and the representation power of deep neural networks [14, 15]. We investigated the usability of such variational meta-RL frameworks for social robotic tasks and realized that the learned encoder suffers from posterior collapse resulting in reduced learning performance (Fig. 2). We propose to use a radial basis function (RBF) layer [16] to transform the task representation before giving it to the downstream task-conditioned policy, thus constructing an embedding that is more suited to representing tasks. RBF networks are universal function approximators [17] whose parameters can be learned and exhibit interesting results in classification tasks [18, 19] as well as in value-learning for continuous action DRL [20].

In summary, our contribution is twofold:

1. We successfully demonstrate the use of off-policy meta-RL on three robotics tasks and four different settings by quickly adapting to various reward functions.
2. We improve the existing PEARL algorithm by introducing an RBF layer that transforms the task representation, enabling better training of the task-dependent behavior.

In this paper, we first discuss other work related to our topic, then introduce our methodology, and finally present the experimental evaluation.

1 Related work

1.1 Adaptive Social Robotics and Reinforcement Learning

Social robotics aims to design robots that share the same space as humans and interact with them in a natural and interpersonal manner. This includes tasks such as human-aware navigation [21] or controlling a robot's gaze to perceive humans optimally [22]. A general overview of social robotics, including methods and their potential application domains, can be found in the following surveys: [23–25]. An important factor for the successful deployment of social robots is their adaptability to different users and social environments. The following surveys provide an introduction to

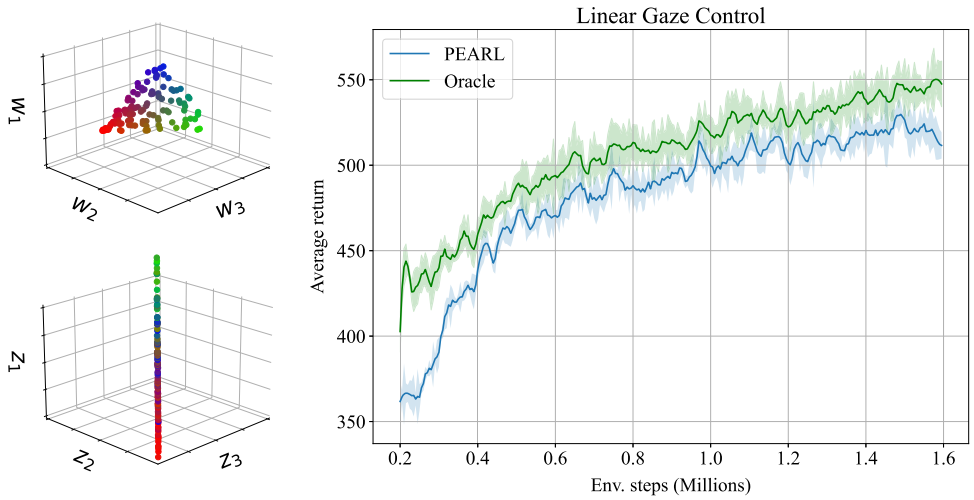


Fig. 2 Posterior collapse in PEARL: (left-top) The ground truth task representation, where each task corresponds to a colored dot. The three primary colors correspond to the strength of each of the three reward component weights. (left-bottom) PEARL's learned task representation uses only one dimension (z_1) and exhibits posterior collapse in the two other dimensions (z_2 and z_3). (right) As a result, PEARL's learning performance compared to a task-conditioned policy using the ground truth (Oracle) is reduced. The proposed RBF-PEARL aims to close this gap by mitigating the negative impact of posterior collapse during training

the field of adaptive social robotics: [26–28]. Adaptive behaviors are generally complex and difficult to engineer by hand, such as with finite state machines or behavior trees [5].

As a possible solution, RL [6] can be used to learn complex social behaviors for robots from direct interactions with the environment. But RL introduces its own challenges as discussed in the survey by Akalin and Loutfi [7] which provides a general overview of the usage of RL in social robotics. One crucial factor for the success of RL is the design of the reward function. The reward function enables the optimal behavior to be learned by giving positive rewards for reaching desired states and behaviors and negative rewards for undesired ones. Given a reward function, RL algorithms attempt to learn the optimal behavior by maximizing the reward.

As the behavior of an RL agent is determined by the reward function, adaptability can be introduced through two approaches: Either 1) by including rewards for adaptability in the reward function, or 2) by adapting the reward function to each social situation. A prominent line of research for the first direction is to obtain rewards directly from a human instructor who teaches the robot. See for example [29], where human feedback is used to learn the optimal proxemics, i.e., how close a robot should position itself to people, for a human-aware navigation controller. While collecting rewards from the user might be suitable for simple tasks requiring few examples to properly learn the task, it is not well suited to more complex learning problems requiring hundreds of samples.

Our work follows the second direction, where social environments and scenarios are modeled by different reward functions. The goal of the RL agent is to adapt quickly to a new reward function. An example of this method is the multitask learning approach in [30], where reward functions are parameterized. The agent learns a task-dependent policy, i.e., a behavior that is conditioned to the parameters of the target reward function. The policy is trained over several reward functions. Given a new task and its reward function parameters, the agent is then directly adapted to it. Nonetheless, this approach is restricted to parameterized reward functions, which are not always available. In this paper, we explore meta-RL methods for adaptive social robotics tasks, which have the advantage of adapting quickly to complex reward functions without the requirement of being parameterized.

1.2 Deep Meta Reinforcement Learning

Deep meta-RL aims to "learn to reinforcement learn," which means to improve the learning speed in a new target task using experience from a set of similar meta-training tasks. The survey by Beck et al. [8] categorizes algorithms by their goal into many-shot and few-shot algorithms.

Many-shot algorithms such as LPG [31], or MetaGenRL [32], try to improve standard RL algorithms by using many observations from the target tasks. This stands in contrast with our goal of adapting quickly to a new social environment.

Few-shot algorithms such as MAML [33] or RL2 [34] aim to adapt to a new target task using only a few observations from that task. MAML learns initialization parameters for a deep network which can then be trained faster in a target task. MAML improves the learning performance, but it uses gradient-based updates to adapt to the target task, still requiring many observations and training iterations for adaptation. In contrast, RL2 uses a recurrent deep network that is trained to identify the task characteristics from observations when executed in its target task, and then to adapt its own behavior to it. RL2 does not require gradient-based updates to adapt and shows improved performance over MAML [13]. A drawback of current MAML and RL2 implementations is that they are on-policy. On-policy methods try to improve the policy that is currently used to collect environment observations [6]. Conversely, off-policy algorithms such as Q-learning can optimize any policy (usually the optimal one) based on the collected observations. This usually makes off-policy methods more sample-efficient than on-policy methods.

For this study, we choose to focus on PEARL [13, 35], an off-policy few-shot meta-RL algorithm. Similarly to RL2, it uses observations from the target task to adapt directly to it without gradient-based updates. Given a target task, PEARL conditions its deep neural model to it with a description of the task in the form of a vector input $z \in \mathbb{R}^d$. It utilizes a variational inference method to learn how to represent tasks and to identify a good task representation based on a few task observations. PEARL has shown that it needs 20 to 100× fewer observations to learn target tasks compared to MAML and RL2 [13], justifying our choice to use it as our base approach for meta-RL in social robotics. We aim to evaluate the use and limitations of PEARL in meta-RL for social robotics and to provide technical solutions to address some of these limitations.

1.3 Posterior Collapse

As for any variational method, PEARL can suffer from posterior collapse, meaning that at least one of the latent bottleneck dimensions becomes non-informative during training. This behavior can be easily identified by looking at the per-dimension Kullback-Leibler divergence between the posterior and prior distributions. Once a dimension collapses, it is not needed to ensure good reconstruction. Several factors can cause posterior collapse, e.g., a local optima [36], the objective function itself [37–40], a too unconstrained variance [41] or the fact that the posterior approximation lags behind the true posterior model [42]. In the case of PEARL, having collapsed dimensions translate into a reduction in the task identification power of the method. To mitigate the negative impact of posterior collapse, we propose to raise the representation power of the latent representation using a radial basis function layer. In our experiments, this proves more beneficial than increasing the network capacity of PEARL.

2 Approach

2.1 Preliminaries

2.1.1 Reinforcement Learning

Tasks in RL are formalized as Markov decision processes (MDPs). An MDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ with state space $\mathcal{S}$ and action space $\mathcal{A}$. The agent transitions from a state $s_t \in \mathcal{S}$ by action $a_t \in \mathcal{A}$ to state s_{t+1} at time step t. The transition probability density function $\mathcal{P}(s_{t+1}|s_t, a_t)$ defines the environment dynamics giving the probabilities for transitions. For each transition, the agent receives a reward defined by the reward function: $r_t = \mathcal{R}(s_t, a_t, s_{t+1}) \in \mathbb{R}$. The probability transition function and reward function are unknown to the agent. The goal of the agent is to maximize the expected future return for each time step t: $G_t = \mathrm{E}\left[\sum_{k=0}^{\infty} \gamma^k \mathcal{R}(s_{t+k}, a_{t+k}, s_{t+1+k})\right]$ where the discount factor $\gamma \in [0, 1)$ defines the importance of short-term rewards relative to long-term ones. RL agents maximize the return by learning a policy $\pi(a|s) = \Pr(A_t = a|S_t = s)$ that defines the probability of the agent taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$.

In the case of robotics tasks, the state is, for example, represented by the input from the robot's visual (cameras) and audio (microphones) sensors. Actions are the motor commands sent to its actuators. Taking an action will have an impact on the environment and will create stochastic change in it. The goal of the task is given by the reward function, e.g., the robot is rewarded if it reaches a certain target position.

2.1.2 Variational Meta-RL (PEARL)

In this work, we would like an agent to adapt quickly to a new reward function using only a few observations. We formalize this problem as a meta-RL problem. Each reward function represents a task $\tau \in \mathcal{T}$ with $\tau = \{\mathcal{R}(s_t, a_t, s_{t+1})\}$. We assume a distribution $\rho(\tau)$ over the task. We differentiate two meta-learning phases: 1) meta-training and 2) meta-testing. During meta-training, a number of training tasks are sampled according to $\rho(\tau)$. Based on these training tasks, a task conditioned policy $\pi(a|s, \tau)$ is learned. During meta-testing, a different set of test tasks is sampled from $\rho(\tau)$ and the adaptation of the learned policy to these tasks is measured.

We chose PEARL [13] as our meta-RL algorithm. PEARL learns a task-conditioned policy $\pi(a|s, z_\tau)$ where $z_\tau \in Z = \mathbb{R}^d$ is a low-dimensional task representation. The task representation z_τ conditions the policy towards maximizing the return of the reward function for task τ. The

representation $z_\tau = f(c^\tau)$ is computed based on observed transitions from a task, called the context $c^\tau = \{c_n^\tau\}$, where $c_n^\tau = (s_n, a_n, r_n, s'_n)$. PEARL uses a variational method, similar to variational autoencoders (VAEs) [12], to estimate the posterior distribution of the low-dimensional representation given the context $p(z|c)$. It approximates the posterior with an inference network $q_\phi(z|c)$ parametrized by ϕ. The network is trained on a log-likelihood objective resulting in the following variational lower bound:

$$\mathbb{E}_\tau \left[\mathbb{E}_{z\sim q_\phi(z|c^\tau)} \left[\mathcal{R}(\tau, z) + \beta D_{KL}(q_\phi(z|c^\tau)) \| p(z)) \right] \right] \quad (1)$$

where $\mathcal{R}(\tau, z)$ is the return for task τ using the policy conditioned on z and $p(z)$ is a standard Gaussian prior over z. The inference network is a product of independent Gaussian factors for each transition in c^τ:

$$q_\phi(z|c^\tau) \propto \prod_{n=1}^{N} \mathcal{N}\left(f_\phi^\mu(c_n^\tau), f_\phi^\sigma(c_n^\tau) \right) \quad (2)$$

where f_ϕ^μ and f_ϕ^σ are represented by a neural network.

During meta-training, the policy $\pi(a|s, z)$ is learned using the soft actor-critic (SAC) [43] algorithm. SAC is off-policy, consisting of an actor $\pi_{\theta_\pi}(a|s, z)$ and a critic $Q_{\theta_Q}(s, a, z)$ network. PEARL jointly trains the inference, actor, and critic networks using the reparameterization trick, similar to VAEs [12]. During a meta-training step, the training procedure has two phases: 1) data collection and 2) network parameter updating. In the data collection step, a replay buffer $\mathcal{B}^\tau$ is filled with the transitions from K trajectories for each training task τ. Then, for each trajectory PEARL samples a task representation $z \sim q_\phi(z|c^\tau)$ to condition the policy where c^τ is sampled from the replay buffer $\mathcal{B}^\tau$. During the second phase, the procedure updates the network parameters for each training task τ. It first samples a batch of context c^τ from recently sampled transitions in the replay buffer $\mathcal{B}^\tau$. Then, the task representation $z \sim q_\phi(z|c^\tau)$ is sampled from the posterior distribution of z given the context c^τ. The critic and actor networks are updated using the task representation and independently sampled transitions from the whole replay buffer. The loss of the critic is given by:

$$\mathcal{L}_{critic} = \mathrm{E}_{\substack{(s,a,r,s')\sim\mathcal{B} \\ z\sim q_\phi(z|c)}} \left[Q_\theta(s, a, z) - (r + \dot{V}(s', \dot{z})) \right]^2 \quad (3)$$

where $\dot{V}$ is the value, i.e., the maximum Q-value, of a target network, and $\dot{z}$ indicates that gradients are not being computed through it. A target network is necessary here because directly implementing Q learning with neural networks was shown to be unstable in many environments [44]. The actor loss is given by:

$$\mathcal{L}_{act} = \mathrm{E}_{\substack{s\sim\mathcal{B}, a\sim\pi_\theta \\ z\sim q_\phi(z|c)}} \left[log(\pi_\theta(a|s, \dot{z}) - (Q_\theta(s, a, \dot{z})) \right] \quad (4)$$

The loss of the inference network for the task representation is composed of the critic loss and the Kullback-Leibler divergence term from (1):

$$\mathcal{L}_\phi = \mathrm{E}_{\substack{(s,a,r,s')\sim\mathcal{B} \\ z\sim q_\phi(z|c)}} \left[\mathcal{L}_{critic} + \beta D_{KL}(q_\phi(z|c^\tau)) \| p(z)) \right] \quad (5)$$

For meta-testing, a test task τ is first explored for a few hundred time steps. The policy $\pi_e(a|s, z_e)$ used for exploration is conditioned to a task representation sampled from the Gaussian prior $z_e \sim p(z)$. After the exploration phase, the context c^τ collected with π_e is used to compute the final task representation given by the sample mean of the posterior mean: $z^\tau = \frac{1}{N} \sum_{n=1}^{N} f_\phi^\mu(c_n^\tau)$. More details on the meta-testing phase are provided in the experimental section (Section 3).

2.2 RBF for Variational Meta-RL

We noticed that in several scenarios PEARL suffers from posterior collapse (of the learned task representation z). As a result, not all dimensions of z are used. The task information is compressed in a few dimensions, making it difficult for the downstream policy and critic network to learn from z. To compensate for this, we propose to transform the task representation $\tilde{z} = \varphi(z)$ to a representation that is easier to process for downstream networks. Inspired by [20], we propose the usage of radial basis function (RBF) layers to transform the task representation.

We construct the RBF layer based on the idea of RBF networks [16], which are universal function approximators [17]. RBF networks consist of a layer of hidden neurons that have a Gaussian activation function. The output of the networks is a weighted sum over the Gaussian activations. In a similar manner, our proposed RBF layer consists of a layer of neurons. For each input dimension $z_j \in \mathbb{R}$ from the original task representation there exist N RBF neurons: $\tilde{z}_{j,1}, \ldots, \tilde{z}_{j,N}$. Each neuron represents a radial basis function having a Gaussian shape:

$$\tilde{z}_{i,j} = \exp\left(-\delta_{i,j} ||z_i - c_{i,j}||^2 \right) \quad (6)$$

where $\delta_{i,j} \in \mathbb{R}$ is a scaling factor and $c_{i,j} \in \mathbb{R}$ is the center, i.e., the point of the highest activation. δ and c are the parameters of the RBF layer, which can be either fixed or trained using gradient descent based on the loss function of downstream networks.

In summary, we propose to transform the task representation using an RBF layer: $\tilde{z} = \varphi(z)$. The resulting representation is then given to the task conditioned actor $\pi_\theta(a|s, \tilde{z})$ and critic $Q(s, a, \tilde{z})$ network (Fig. 3).

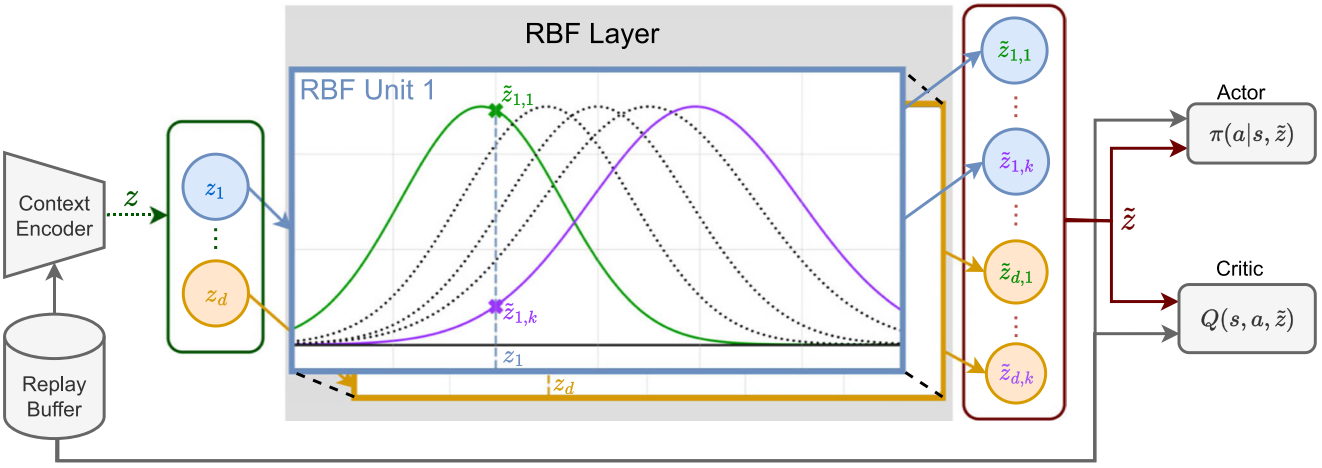


Fig. 3 PEARL-RBF Architecture: The proposed meta-RL procedure learns to adapt to unseen test reward functions. The context encoder uses data from the replay buffer to infer the posterior over the latent context variable z. The latent context sampled from the posterior $z = (z_1, \dots z_d)$ is then fed to the RBF network, which uplifts every input dimension m to a different k-dimensional representation: $z_m \rightarrow (\tilde{z}_{m,1}, \dots, \tilde{z}_{m,k})$. The resulting task representation $\tilde{z}$ is used to condition the actor and critic network

3 Experimental Results

We evaluated our approach in three different environments inspired by social interaction scenarios. In the first environment, the agent, represented by a robotic head, needs to learn how to control its position to optimize the number of people in its field of view. In the second environment, the agent must learn to navigate safely and in a socially compliant manner through a crowd of people. The third environment is a continuous control environment focused on robotic locomotion. The purpose of these three environments is twofold. First, we want to show the effectiveness of our proposed methodology to generate different behaviors in human-robot interaction tasks. Second, we assess whether or not the use of RBF layers is beneficial to variational meta-RL in terms of training and adaptation efficiency.

3.1 Evaluation protocol

3.1.1 Baselines

A natural baseline to the proposed RBF-PEARL is standard PEARL. However, directly comparing with PEARL seems unfair, since adding RBF layers increases the number of parameters of the actor and critic networks. We therefore adjust the number of parameters of the actor and critic networks of the PEARL baseline to match the number of parameters of the RBF-PEARL. Beyond this adjustment, both methods use separate actor and critic networks consisting of a 3-layer MLP with 300 neurons per layer. Both methods use a 3-layer MLP with 200 neurons per layer as the task encoder. In more detail, if the latent dimension is d, and we use one RBF layer with k neurons, the first layer of the actor/critic networks would have $300d$ parameters for PEARL and $300kd$ for RBF-PEARL. For a fair comparison, we add an extra layer at the beginning of PEARL's actor/critic network with kd output neurons, rendering the number of PEARL's and RBF-PEARL's parameters comparable. Unless otherwise stated, we use $k = 9$.

As a separate baseline, we used a modified version of the soft-actor-critic algorithm. We trained one agent per task, using only 200 observations. However, we strongly increased the number of gradient steps performed per observation in order to force the network to learn a behavior with only these 200 observations. We used this baseline to compare the performances of classical RL algorithms (here SAC) trained on 200 observations with respect to RBF-PEARL and PEARL with 200 adaption steps.

We also compared the above algorithms against two state-of-the-art on-policy meta-RL methods:

- MAML-PPO, an on-policy gradient-based meta-RL algorithm that embeds policy gradient steps into the meta-optimization, and is trained with PPO [33],
- RL2, an on-policy meta-RL algorithm that corresponds to training a GRU network with hidden states maintained across episodes within a task, trained with PPO [34].

For these two baselines, we used the implementation provided by the Garage reinforcement learning library [45], a toolkit used to evaluate meta-RL and RL methods, providing state-of-the-art implementations. Both of these baselines

are evaluated on the same number of environment steps as PEARL.

3.1.2 Procedure

All three environments are evaluated with the same procedure. For meta-training, we sample 100 tasks. Evaluation is performed on 20 meta-testing tasks that are different from those used during meta-training. For PEARL and RBF-PEARL we collect observations for 200 time steps with a task representation z_e sampled from the standard Gaussian prior on each meta-testing task τ. The 200 observations are then used to compute the task representation z^τ sampled from the posterior estimation given by the encoder. To compute the final test-time performance, we record the performance of the policy associated with the task representation z^τ for one episode. We repeat the training process 5 times and report the mean performance and the associated standard error.

3.2 Gaze control environment

3.2.1 Environment description

The goal in the gaze control environment is to learn a gaze strategy for a robotic head. The environment is based on the work by Lathuiliere et al. [22]. The robot's observations are multimodal, consisting of visual, auditory, and proprioception cues. The entire visual scene has a size of 2×1. The agent observes the scene with a head camera that extracts pose cues from a field of view (FOV) of the size 0.4×0.3. The pose cues are represented by a heatmap for each of $J = 18$ landmarks such as for the nose, neck, left shoulder, or right hip of people. The heatmap associated with each landmark indicates the probability of that landmark being present at every position. Since state-of-the-art pose estimators provide these heatmaps in low resolution, the visual observations will consist of J 7×7 heatmaps. Regarding the auditory observations, we emulate the output of a sound source localization algorithm. Given the precision of current sound source localization methods, it seems reasonable to represent the auditory input as a 14×8 heatmap corresponding to the entire scene. Each of its cells corresponds to the probability of having an active speaker in this direction. Importantly, while the visual features correspond to the current FOV of the camera, the auditory features correspond to the entire scene, since audio localization is not limited by the camera's FOV. Lastly, the proprioception cues consist of the robot's head orientation, i.e., the coordinates of the robot's current view center. It is encoded using a $\mathbb{R}^2$ vector representing both the pan and tilt angles of the robotic head. An observation of the gaze control environment is shown in Fig. 4 (left).

The environment has a two-dimensional action space $[-1, 1]^2$, corresponding to the pan and tilt angular velocities. We choose to normalize the action space to stabilize the network learning. The maximum pan and tilt velocities correspond to shifting the camera FOV by 0.16 and 0.11 in scene coordinates at every time step.

3.2.2 Reward components

We define three reward components that will be combined to generate various tasks (i.e., reward functions). We will focus on the number of people in the FOV, the presence of a speaker in the FOV, and reducing spurious robot movements. The dimension of the latent space of both PEARL and RBF-PEARL is set to $d = 3$.

We want to reward the robot for having people in the FOV of the camera since this means that the robot would be looking at people. Naively rewarding the number of people within the FOV leads to an action policy that explores until it finds a person, and then follows this person. A more natural behavior is to check on previously detected people. We therefore propose to use a visual reward component that depends on the last time the agent saw a person. More precisely:

$$R_{\text{vis}} = \sum_{p \in P_{\text{vis}}} 2 - \exp(-t_p) \tag{7}$$

where P_{vis} is the set of people whose face is in the visual FOV and $t_p \in [0, \infty]$ is the time since the person was last seen (before the current frame). When a person remains in the FOV, the reward is close to 1. When a person not seen for a long time reappears in the FOV, the reward is close to 2. In this way, the desired behavior is encouraged.

As well as maximizing the number of visible people, we would like the robot to preferentially look at the speaking person(s). Therefore, we define the audio component of the

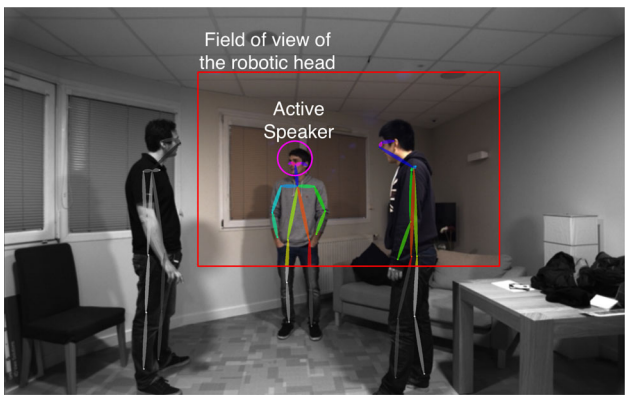


Fig. 4 Illustration of the gaze control environment: The FOV is shown in red, the active speaker in purple, and the visible/non-visible landmarks are shown in white/color respectively

reward function as:

$$R_{\text{aud}} = \begin{cases} 0 & \text{Nobody speaks} \\ -0.5 & \text{Speakers are outside the FOV} \\ 2|P_{\text{aud}}| & \text{Speakers within the FOV} \end{cases} \quad (8)$$

where P_{aud} is the set of speakers in the FOV.

Finally, we would like to penalize expansive and brusque movements, since they are quite unnatural in social interactions. We define a negative movement component of the reward function, defined as:

$$R_{\text{mov}} = -K_{\text{mov}}\sqrt{a_{pan}^2 + a_{tilt}^2} \quad (9)$$

where a_{pan} and a_{tilt} are the two components of the action space, and $K_{\text{mov}} = 16$ is a constant to put R_{mov} in a similar numeric range to R_{vid} and R_{aud}.

3.2.3 Tasks

In order to construct different tasks (reward functions), we propose to use the defined components in two different ways. First, with simple combinations as in [22], and then with more complex ones (i.e., non-linear). The first family of reward functions is generated by sampling convex combinations of the three components defined above:

$$R^{\tau} = \omega_{\text{vis}}^{\tau} R_{\text{vis}} + \omega_{\text{aud}}^{\tau} R_{\text{aud}} + \omega_{\text{mov}}^{\tau} R_{\text{mov}} \quad (10)$$

where $\omega_{\text{vis}}^{\tau}, \omega_{\text{aud}}^{\tau}, \omega_{\text{mov}}^{\tau} \in [0, 1]$ are random convex weights, meaning that $\omega_{\text{vis}}^{\tau} + \omega_{\text{aud}}^{\tau} + \omega_{\text{mov}}^{\tau} = 1$.

We also wanted to compare the algorithms in more complex environments. To that end, we design our second family of reward functions using random multi-layer-perceptron (MLP) networks. These input the value of the three reward components defined above. The MLPs have 1 to 3 layers with 4 to 6 neurons each. They have either no activation function with probability 0.25 or a sigmoid with probability 0.75. The number of layers, the neurons per layer, the activation, and the weights are sampled randomly. Formally, we write:

$$R^{\tau} = f^{\tau}(R_{vis}, R_{aud}, R_{mov}; W^{\tau}) \quad (11)$$

where f^{τ} represents the sampled MLP network with sampled connection weights W^{τ}.

3.2.4 Results

We report the average return over the set of meta-testing tasks over the meta-training iterations (Fig. 5). More precisely, we plot the average return mean and standard deviation over the five independent runs, for the gaze control environment with convex (left) and non-linear (right) combinations of reward components.

Generally, both PEARL and RBF-PEARL are able to provide a better adaptation starting point with the training progress. In addition, we observe that RBF-PEARL has a steeper learning curve than PEARL on both types of reward functions. More precisely, for convex combinations of reward components, RBF-PEARL performs comparably to PEARL during the first 400k steps. From this point on, RBF-PEARL systematically outperforms PEARL by a margin of 20-30. For the non-linear combinations of reward components, RBF-PEARL exhibits superior performance from 600k meta-training steps on, by a margin of roughly 50. Overall, in these two families of tasks, RBF-PEARL is faster than PEARL and shows a better asymptotic perfor-

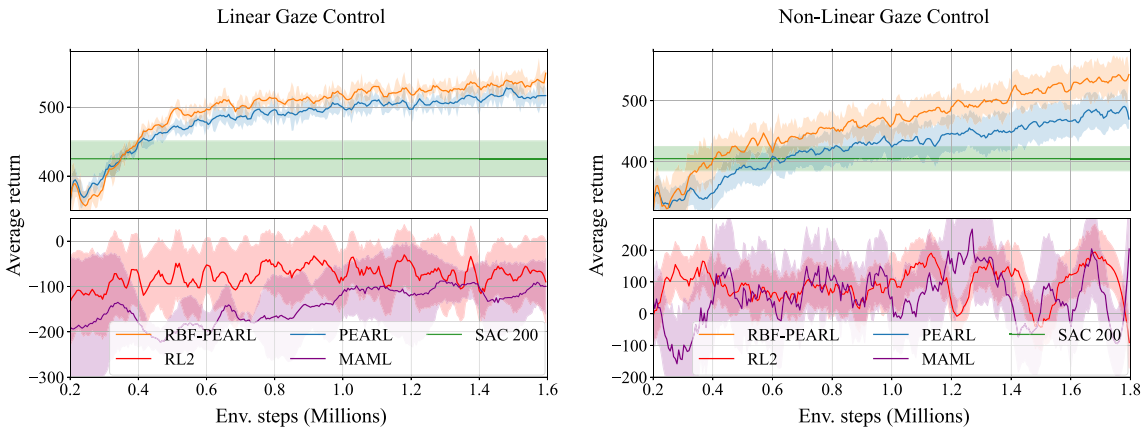


Fig. 5 Results for the gaze control environment: RBF-PEARL outperforms PEARL by 30 and 70 in terms of average test-task return. We plot the test-task performance (average return) vs. the number of collected samples during meta-training on the linear gaze control (left) and the non-linear gaze control (right) environments. In both cases, the off-policy meta-RL methods outperform SAC 200, which in turn outperforms on-policy meta-RL methods

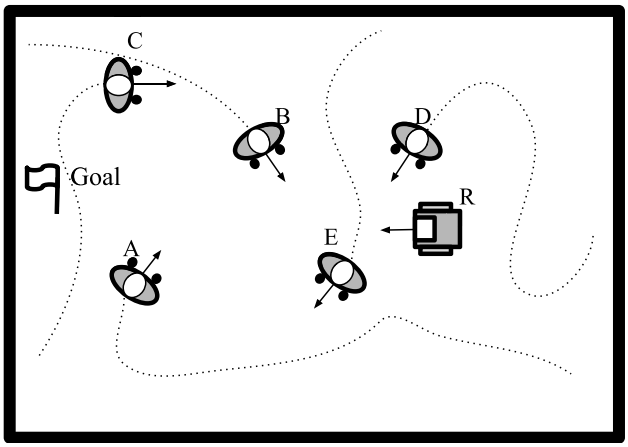


Fig. 6 Schematic representation of the social navigation environment: The robot R must move towards its goal position while navigating around five human agents ($A, \ldots, E$)

mance, thus showcasing the benefit of using the RBF layer. As expected, both PEARL and RBF-PEARL achieve better performances than the batch of agents trained using the soft actor-critic algorithm with only 200 observations (SAC 200). In the Linear Gaze Control environment, the average performance achieved by agents trained with the soft-actor-critic algorithm is inferior to the performances of PEARL and RBF-PEARL by a margin of 100. In the non-linear environment, the gap is lower between PEARL and SAC 200 as PEARL outperforms SAC 200 by a margin of 50. With RBF-PEARL, the performance gap with SAC 200 is more significant, as RBF-PEARL exhibits a superior performance by a margin of 130.

We observe that PEARL significantly outperforms on-policy meta-RL baselines in both tasks, as shown by the final performances after 1.6 and 1.8 million environment steps. More precisely, in the linear environment, the average performance of MAML is -99, and the average performance of RL2 is -89. In the non-linear environment, the average performance of MAML is 203, and the average performance of RL2 is -90. These poor performances are most likely due to the poor sample efficiency of on-policy methods.

3.3 Social navigation environment

3.3.1 Environment description

In this environment, the agent has to learn a human-aware navigation strategy [21]. Our simulation environment is an empty room of dimension 15 × 10 m (Fig. 6). The room is populated with five human agents, whose position at the beginning of each episode is randomly initialized. Likewise, we randomly sample a robot target position at the beginning of an episode. The robot should reach the target position before the end of the episode without disturbing the human agents. Each human agent is given a random target position that they will go towards. If the robot reaches its goal it will be assigned a new one. To simulate the behavior of human agents, we model the human agents' motion using a social force model [46] to generate plausible trajectories. This framework also limits the number of collision between human agents.

The robot agent has access to the coordinates of all people in the scene, their velocities, and their orientations. The robot also has access to its own velocity, coordinates, and target position. While the robot always begins the episode at the same position (coordinates $(14, 5)$), its target position is randomly sampled following $x \sim \mathcal{U}(0.0, 2.0)$ and $y \sim \mathcal{U}(0.0, 10.0)$, for the x and y coordinates, respectively.

The action space is a continuous two-dimensional space set to $[-15, 15] \times [-2, 2]$, which corresponds respectively to the angular velocity in $rad.s^{-1}$ and linear velocity in $m.s^{-1}$. The maximum value of the linear and angular velocity is chosen such that the robot can go as fast as any human agent in the scene. In this environment, we use a smaller number of neurons $k = 5$ for our RBF layer.

3.3.2 Reward components

We define a set of five reward components that will be combined to generate various tasks. The dimension of the latent space of both PEARL and RBF-PEARL is set to $d = 5$ for this environment.

First, the goal component R_g rewards the agent for reaching the target position:

$$R_g = 1 - \frac{d(r, g)}{D} \tag{12}$$

where $d(r, g)$ is the distance between the robot and the goal and D is a normalizing factor to guarantee that the goal component stays within $[-1, 1]$.

Second, the collision component R_c penalizes collisions between the robot and human agents:

$$R_c = \begin{cases} 0 & d(r, h_i) > d_c \\ -1 & d(r, h_i) < d_c \end{cases} \tag{13}$$

where $d(r, h_i)$ is the distance between the robot and the human agent i and d_c is the collision threshold between the robot and the human agent.

Third, the social component R_s rewards the robot for maintaining a safe distance away from all human agents. The social component depends on the distance between the robot and each human agent. If the distance between the robot and one of them is below a certain threshold, the robot will be

penalized. The closer the robot is to a person, the higher the penalty will be. If the robot is close to more than one human agent, only the closest person to the robot is taken into consideration (that is, the human agent that will generate the lowest reward):

$$R_s = \min_i \left[\frac{d(r, h_i)}{d_s} - 1 \right] \tag{14}$$

where $d(r, h_i)$ is the distance between the robot and the human agent i and d_s can be understood as a threshold. Indeed, if the minimum distance between the robot and a human is below d_s, the reward becomes negative. Thus, d_s can be seen as the distance at which the robot enters the comfort zone of people.

Fourth, the approach component R_a is designed to reward the robot for positioning itself to avoid making other human agents aware of it. This component is based on the approach by Satake et al. [47], who quantify how aware people are of a robot when it approaches them. The awareness is based on the relative positions and orientations of the robot and the human agents. We compute the robot's awareness of each human agent and reward a low awareness of the robot. Awareness is a combination of visibility $R_{\text{visible},i}$ and direction $R_{\text{direction},i}$. While visibility assesses how visible the robot is to the human agent, direction assesses if the robot is going in a direction that would make the human agent more aware/afraid of it.

Visibility for human agent i is defined as:

$$R_{\text{visible},i} = \begin{cases} 1 - \frac{\theta_{i,r}}{\theta_{th}} & \theta_{i,r} < \theta_{th} \\ -\frac{\theta_{i,r} - \theta_{th}}{\pi - \theta_{th}} & \text{otherwise} \end{cases} \tag{15}$$

where θ_{th} is a threshold angle from which the robot is visible to the human agent and $\theta_{i,r}$ is the angle of the robot relative to the i's human agent motion direction. If the robot is in front of the human agent, the value will be close to 1, and if the robot is behind the human the value will be close to -1.

The direction for human agent i is defined as:

$$R_{\text{direction},i} = \begin{cases} 1 - \frac{\theta_{i,r}}{\pi/2} & \theta_{i,r} < \theta_{th} \\ 1 & \text{otherwise} \end{cases} \tag{16}$$

It is designed to address how the human agent perceives the robot coming toward him. If the robot is coming closer to the human agent, he/she will become more aware of it and be distracted by it. On the other hand, even if the robot is visible to the human agent, if it moves away from him/her, the human agent will be less likely to be distracted by the robot.

To compute the approach reward component, we compute the minimum over the visibility and direction product over the human agents:

$$R_a = \min_i R_{\text{visible},i} \cdot R_{\text{direction},i} \tag{17}$$

Fifth, and last, the velocity component R_v is designed to penalize the robot for going too fast when it is in front of people. It is preferable to avoid having the robot moving fast in front of people as they may be disturbed or even afraid of it. The velocity component therefore penalizes the robot for moving fast if it is visible to human agents i:

$$R_{v,i} = \begin{cases} -e^v(1 - \frac{\theta_{i,r}}{\theta_{th}}) & \theta_{i,r} < \theta_{th} \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

where θ_{th} is a threshold angle from which the robot is visible and $\theta_{i,r}$ is the angle of the robot relative to the human agent's motion direction. To compute the reward component, we adopt the same methodology as with the social and approach component. We choose the minimum value obtained from all the human agents in the scene:

$$R_v = \min_i R_{v,i} \tag{19}$$

3.3.3 Tasks

We evaluate the meta-RL algorithms on reward functions that are convex combinations of the five reward components:

$$R^\tau = \omega_g^\tau R_g + \omega_c^\tau R_c + \omega_s^\tau R_s + \omega_a^\tau R_a + \omega_v^\tau R_v \tag{20}$$

where $\omega_g^\tau, \omega_c^\tau, \omega_s^\tau, \omega_a^\tau, \omega_v^\tau \in [0, 1]$ are the randomly sampled convex weights leading to task τ, meaning that $\omega_v^\tau + \omega_c^\tau + \omega_s^\tau + \omega_a^\tau + \omega_v^\tau = 1$. The social navigation environment is more challenging than the gaze control environment. The higher number of reward components makes the learning process more difficult, as well as behavior generation as there is a wider variety of possible reward functions.

3.3.4 Results

We report the average return over the meta-testing tasks with the progress of meta-training for PEARL and RBF-PEARL (Fig. 7). As in the previous environment, we report mean and standard deviation over five runs. Similarly to the previous case, the performance of the two methods looks similar during the first steps of the training. More importantly, the RBF layer seems to have a positive impact on the asymptotic performance. Indeed, after 6 million steps, the RBF-PEARL

algorithm performs better than PEARL. Similarly to the gaze control environment, both PEARL and RBF-PEARL perform better than SAC-200 by a large margin (170). They also outperform both on-policy methods, MAML and RL2, by a margin of 304 and 164, respectively.

3.4 Racer environment

3.4.1 Environment description

We evaluated the algorithms in an additional non-social task with complex, non-linear reward functions, called the racer environment [48]. The agent has to navigate in a continuous two-dimensional scene for two hundred time steps (Fig. 8). Similar to a car, the agent has an orientation and momentum, so that it can only drive straight, or around a right or left curve. The agent reappears on the opposite side if it exits one side. At the beginning of an episode, the agent is randomly placed in the environment. The agent's state is a vector $s \in \mathbb{R}^{120}$ corresponding to the agent's position and orientation. The position is encoded using a 10×10 evenly distributed grid of two-dimensional Gaussian radial basis functions. Similarly, the orientation is also encoded using 20 Gaussian radial basis functions. The action space of the agent consists of a one-dimensional continuous space set to $[-1, 1]$. The value of the action corresponds to the force applied to the agent, which then modifies the agent's orientation and position. For example, if the value of the action is close to -1, the agent will make a left curve.

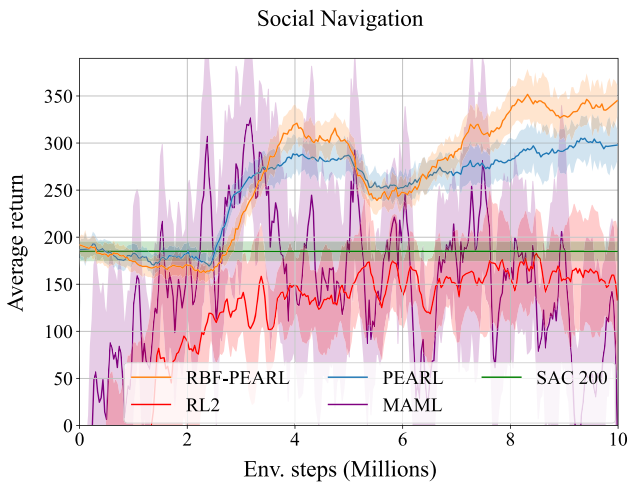


Fig. 7 Results on the social navigation environment: PEARL-RBF outperforms PEARL-RBF in average test-task performance by 40. We plot the average test-task return vs. the number of samples collected during meta-training on the social navigation environment. Both off-policy meta-RL methods outperform SAC 200, which in turn outperforms both on-policy meta-RL methods

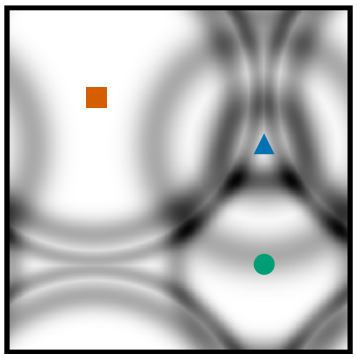


Fig. 8 Illustration for the racer environment: The three markers are depicted in blue, green, and orange. Dark regions correspond to high-reward regions. In the task depicted in the figure, the red and blue markers have two Gaussians, while the green one has only one. The number of Gaussians, their mean, and standard deviation are randomly sampled for each task

3.4.2 Reward components

We define three reward components, each of them associated with one of the three markers in Fig. 8. More precisely, each reward component r_k is defined as the maximum over Gaussian-shaped functions over the distance to the k-th marker d_k:

$$r_k = \max \left\{ \exp\left(-\frac{(d_k - \mu_{k,j})^2}{\sigma_{k,j}} \right) \right\}_{j=1}^{n_k} \tag{21}$$

where n_k is the number of Gaussians for marker k, and $\mu_{k,j}$ and $\sigma_{k,j}$ are the mean and standard deviation of the j-th Gaussian of marker k. For each task, the parameters of the reward components are randomly sampled, as explained below.

3.4.3 Tasks

The tasks differ in terms of the parameters of each of the three reward components. The number of Gaussians is sampled uniformly: $n_k \sim \mathcal{U}\{1, 2\}$. The two parameters of each Gaussian component are sampled according to $\mu_{k,j} \sim \mathcal{U}(0.0, 0.7)$ and $\sigma_{k,j} \sim \mathcal{U}(0.001, 0.01)$. This sampling instantiates the three reward components for task τ, r_k^τ, and the final reward function is written:

$$R^\tau = \frac{1}{3} \sum_{k=1}^{3} r_k^\tau(d_k) \tag{22}$$

The dimension of the latent space of both PEARL and RBF-PEARL is set to $d = 3$.

3.4.4 Results

RBF-PEARL consistently shows a stronger performance than PEARL in the racer environment (Fig. 9). After 300k steps, RBF-PEARL constantly outperforms PEARL. The asymptotic performance of RBF-PEARL is also higher than PEARL. The average return at the end of the training is $54 \pm (11)$ for PEARL and $65 \pm (2)$ for RBF-PEARL. Also, both PEARL and RBF-PEARL (56 and 65 respectively) outperform SAC 200 by a large margin, performing two times better: SAC 200 obtains a performance of only 30. Once again, we notice that PEARL outperforms on-policy meta-RL baselines. The final performance of MAML and RL2 after 1.1 million environment steps is 51 and 48, respectively, far below the performance of PEARL for the same number of environment steps.

4 Discussion

We discuss six questions that we believe deserve some attention. First, whether variational meta-RL is well suited to social robotics. Second, what the difference is between the off-policy vs. on-policy approaches. Third, the impact of the RBF layer in variational meta-RL. Fourth, how the trainability of the RBF layer's parameters and the number of RBF neurons influence its performance. Fifth, what mechanisms are behind the improved performance of the RBF layer. And last, open research directions in the field of meta-RL for social robotics.

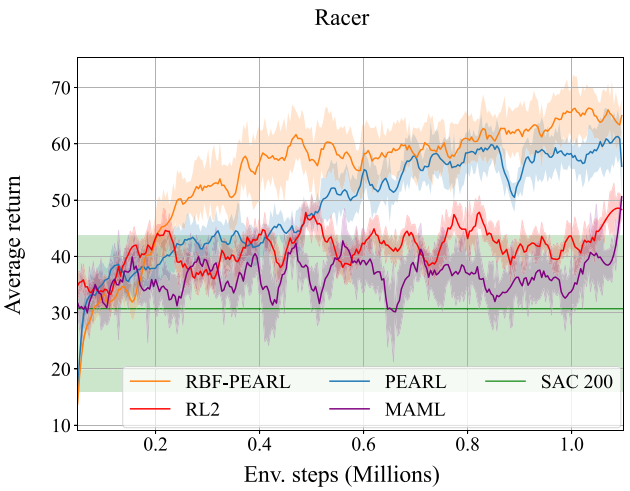


Fig. 9 Results on the racer environment: PEARL-RBF outperforms PEARL in average test-task return by 10. We plot the average test-task performance vs. the number of samples collected during meta-training. Off-policy meta-RL methods outperform on-policy meta-RL methods and SAC 200. It is the only environment where on-policy meta-RL outperforms the SAC 200 baseline

4.1 Variational Meta-RL for Social Robotics

We proposed the application of variational meta-RL to allow robots to quickly adapt to different social scenarios. We achieved this by enabling robots to adapt to new reward functions which define the requirements of social scenarios, such as humans' different preferred social distances. Indeed, our results from four simulation experiments show that with a variational meta-RL procedure (PEARL), robotic agents are able to quickly adapt to different scenarios, i.e., reward functions, requiring only 200 observations (Figs. 5, 7, and 9). Conversely, a classical RL algorithm trained on 200 observations (SAC 200) achieves a significantly lower performance. In the linear and non-linear gaze control environments, the performances of SAC 200 were respectively 23% lower and 25 % lower than those of RBF-PEARL. In the social navigation environment, the performances of SAC 200 were 47% lower. In the racer environment, the performances of SAC 200 were 53% lower than those of RBF-PEARL.

To further demonstrate the interest of variational meta-RL, we compared the performance of SAC trained on more than 200 steps with the final models obtained with PEARL and RBF-PEARL. As in the previous experiments, the two meta-RL algorithms only need 200 environment steps to adapt to each test environment. Experiments were carried out for the linear-gaze, non-linear gaze, and racer environments (Fig. 10). It should be noted that for these experiments, the number of gradient descent iterations per collected environment observation is the same for the meta-RL algorithms and for SAC. In the previous experiments (Sec. 3), SAC was given more gradient descent iterations per observation to allow it to converge to a policy. In the first two environments, SAC requires 50k environment steps to reach the performance of PEARL and RBF-PEARL, which only benefit from 200 environment steps. Surprisingly, even after training for 200k steps, SAC does not significantly outperform the two meta-RL algorithms. This is especially true for the third environment (racer), where the performance of SAC does not seem to be superior to the meta-RL algorithms, even after 100k environment steps. It would appear that SAC does not manage to learn a moderately optimal action policy in the racer environment. After looking carefully at our results, we realize that the optimal hyperparameters of SAC depend strongly on the learned task, and no set of parameters seems to be commonly optimal for all tasks.

Lastly, we evaluated whether the meta-policy learned by RBF-PEARL is able to produce diverse and meaningful behavior when adapted to different reward functions. We plotted the trajectories of the robot agent in the social navigation environment for four different reward weight combinations (Fig. 11). The meta-policy is adapted to each combination based on observations from 200 time steps. The adapted behaviors are easily distinguishable from each other.

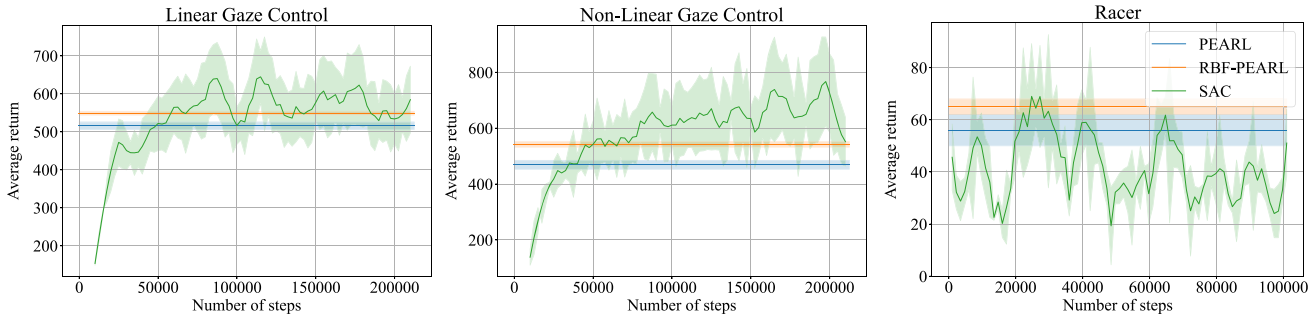


Fig. 10 Performances achieved by PEARL and RBF-PEARL compared to full training with SAC: SAC needs more than 100× environment steps: In order to reach the performance of PEARL/PEARL-RBF, SAC needs several thousand environment steps, compared to the 200 environment steps needed by PEARL/PEARL-RBF after the meta-training phase. Results show the average test-task performance and standard deviation per environment step over 20 tasks and 5 seeds. (left): Gaze control environment with linear reward functions. (middle): Gaze control environment with non-linear reward functions. (right): Racer environment. The sensitivity of SAC to the hyperparameters is the main cause of its low performance on the racer environment, while PEARL/PEARL-RBF do not suffer from this adverse effect

For a reward function that depends only on reaching the goal (a), the robot has trouble reaching the target position as it bumps into humans twice, and does not manage to modify its path accordingly. However, weighting the speed component (b) more helps the robot to reach the target position, as the robot learns to have better control over its angular and linear speed. By weighting the social and approach components (c and d) more, the robot actively tries to avoid people around it, even if this results in a failure to reach the target position. In conclusion, variational meta-RL successfully and quickly adapts an agent to different reward functions, allowing it to efficiently find appropriate behaviors for different social scenarios.

4.2 Sample efficiency of off-policy meta-RL

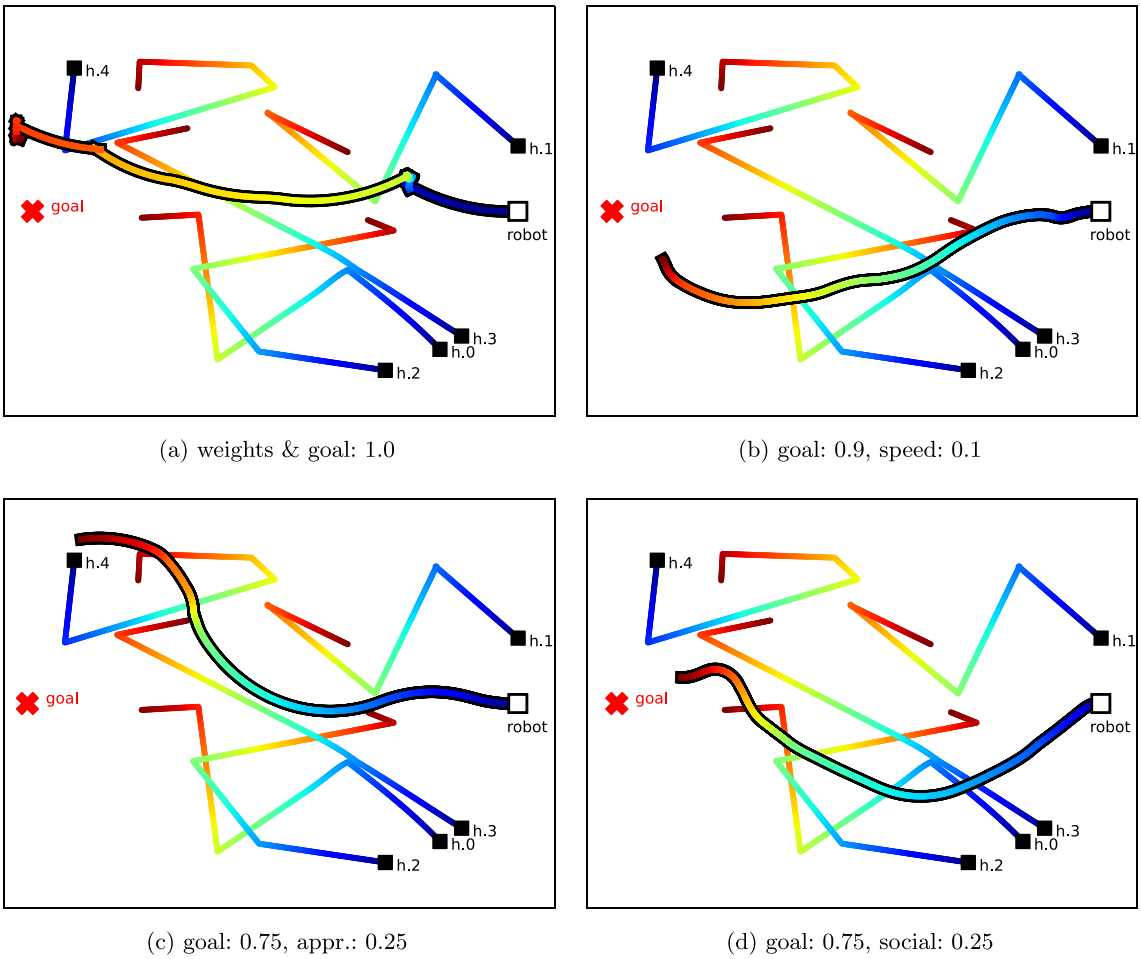
We compared the performances of off-policy meta-RL, such as PEARL, with other on-policy methods proposed in the literature, such as MAML and RL2. We find that PEARL significantly outperforms on-policy meta-RL methods across all domains. PEARL converges to its final asymptotic performance with 3 to 10 times fewer samples during meta-training than other approaches proposed in the literature. Increasing the sample efficiency is generally speaking positive, specially for robotics applications. Indeed, running large amounts of environment steps in robotics applications is time-consuming, expensive, and requires significant human efforts. Fewer samples reduce the cost and effort required for data collection and labeling, making it more feasible to develop and deploy robotics systems in the real world.

However, the actor-critic architecture used in models such as PEARL is associated with higher training complexity compared to models like MAML and RL2, as it requires the tuning of a greater number of hyperparameters. Also, PEARL may face more difficulties in generalizing to unseen environments, as the embedding may fail to capture the relevant information required for effective adaptation. Our proposed approach, RBF-PEARL, tends to mitigate this latter issue by transforming the task representation with our radial basis function (RBF) layer, and to improve performances compared to PEARL.

4.3 Performance of PEARL vs. RBF-PEARL

We compared the performance of RBF-PEARL to two versions of PEARL. The first version, called PEARL, uses a neural network model with a similar number of parameters to RBF-PEARL. This is achieved by using an MLP layer with kd output neurons, where k is the number of neurons of the RBF layer and d is the latent dimension, followed by a ReLU activation unit instead of an RBF layer before the actor and critic network to process the task representation z. All four experiments show that RBF-PEARL consistently outperforms PEARL in meta-test task performance (Figs. 5, 7, and 9). We further compared their asymptotic performances in the 20 meta-testing tasks in more detail (Table 1). In the two linear environments, RBF-PEARL outperforms PEARL by a margin of 6% in the linear gaze control environment and 16% in the social navigation environment. For the two non-linear environments the improvement is larger. In the racer environment, the performance is 16% better than that obtained by PEARL. In the non-linear gaze control environment, the difference is 18% compared to PEARL.

In addition, we report results obtained with a version of PEARL without the additional MLP layer, called Vanilla PEARL. We find that the performances of PEARL and Vanilla PEARL are very similar in three of the four tested environments (Table 1). The difference in the average final asymptotic return between the two is less than 5%, and is



(a) weights & goal: 1.0

(b) goal: 0.9, speed: 0.1

(c) goal: 0.75, appr.: 0.25

(d) goal: 0.75, social: 0.25

Fig. 11 Trajectories in the social navigation environment for 4 different weight combinations of the reward function: RBF-PEARL learns to generate different behaviors depending on the reward weight combination. The robot's trajectory is represented by the line with the black contour. The color indicates the time step

within their confidence ranges. PEARL is only significantly better than Vanilla PEARL (by 18%) in the social navigation environment.

In summary, the results of PEARL and Vanilla PEARL

Table 1 Final performance of PEARL, RBF-PEARL and on-policy baselines

Algorithm	Lin. Gaze	Non-Lin. Gaze	Social Nav.	Racer
PEARL	516 ± 10	460 ± 16	297 ± 30	56 ± 6
Vanilla PEARL	514 ± 6	469 ± 16	253 ± 18	56 ± 5
RBF-PEARL	$\mathbf{549} \pm 5$	$\mathbf{542} \pm 10$	$\mathbf{344} \pm 31$	65 ± 3
RBF-PEARL-fp	543 ± 7	518 ± 15	322 ± 12	$\mathbf{72} \pm 6$
MAML	-99 ± 61	203 ± 189	-7 ± 97	51 ± 1
RL2	-89 ± 90	-90 ± 81	133 ± 74	48 ± 2

RBF-PEARL outperforms PEARL and the learning of RBF parameters is in most environments beneficial over fixed parameters (RBF-PEARL-fp). On-policy algorithms (MAML and RL2) struggle to learn good policies with the same number of environment steps. Reported are the mean ± standard deviation over 5 seeds for each algorithm meta-testing performance at the end of meta-training

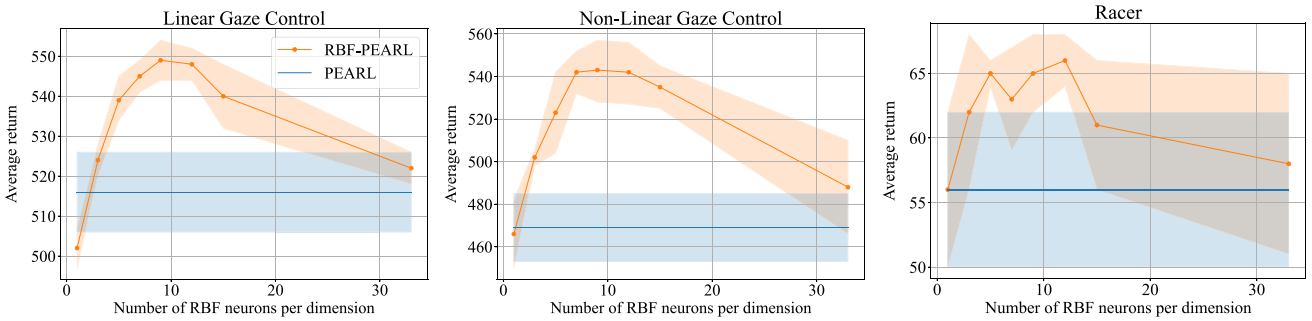


Fig. 12 Ablation study on the number of RBF neurons per input dimension for RBF-PEARL: The optimal number of neurons is around 10 for the three evaluated environments. Reported are the mean and standard deviation over 5 seeds for each algorithm meta-testing performance after a certain number of meta-training steps

compared to RBF-PEARL show that the RBF layer improves the performance of PEARL significantly. This effect cannot be explained by a difference in their model capacity, as PEARL and RBF-PEARL have a similar number of parameters. Instead, the computational properties of the RBF layer seem to be the important factors.

4.4 Impact of trainability and number of RBF neurons

We evaluated the effect of training the RBF parameters, i.e., centers c and scaling factors δ (6), to an RBF-PEARL architecture with fixed parameters (RBF-PEARL-fp). The centers are fixed by evenly distributing them over an interval that was set to encompass the space of task representations z. The scaling factors are fixed based on a function of the distance between center points. They were chosen so that two neighboring RBF neurons both have an activation of 0.5 for a representation $z_k \in \mathbb{R}$ lying in the middle between their centers. Overall, we see that training the centers and scaling factors of the RBF layer has a beneficial impact on the asymptotic performances of the RBF-PEARL algorithm (Table 1). In the non-linear gaze control and social navigation environments, training the RBF parameters improves the final asymptotic performances by 2% to 6%. Only in the racer environment does the fact of having fixed parameters improve performance by 10%. In summary, the advantage of learning the parameters of the RBF layer is task-dependent, but seems to be beneficial to most tasks.

Lastly, we analyze the effect of the number of neurons per input dimension in the RBF layer (Fig. 12). In the three evaluated environments, we found that the number of neurons has a noticeable effect on the asymptotic performance. The optimal number is around 10 in the three evaluated environments: 9 for linear gaze control, 9 for non-linear gaze control, and 12 for the racer. We believe that the drop in performance for higher numbers of neurons may be due to overfitting on the training tasks.

4.5 How does the RBF layer improve performance?

The RBF layer improves the performance of the variational meta-RL procedure, but what are the mechanisms behind this improvement? In general, the layer can have three potential influences on the learning procedure. First, as its input, task representation z is also learned, and could alter the learning objective of z, resulting in a different representation. Second, it could alter the temporal learning dynamics of the task representation, also leading to different learning dynamics in the downstream policy and value networks. Third, its output $\tilde{z}$ could provide an improved representation of the policy and value networks. We investigated these factors in the linear gaze control task (Sec. 3.2). We restricted our analysis to an RBF layer with the 3 RBF neurons per input dimension. This low-dimensional representation makes it easier to analyze and visualize the results compared to the optimal configuration with the 9 RBF neurons per input dimension.

4.5.1 Influence on the learned input task representation *z*

The learned task representations z of PEARL without (Fig. 2, left-bottom) and with an RBF layer (Fig. 15) have only minor differences. The average and standard deviation over the 100 meta-training task representation mean that μ per dimension are for PEARL: z_1: 0.041 ± 3.676 , z_2: 1.013 ± 0.006, z_3: 1.009 ± 0.003; and for RBF-PEARL: z_1: 0.061 ± 3.507, z_2: 1.018 ± 0.009, z_3: 1.033 ± 0.032.

Both representations have a posterior collapse in two (z_2, z_3) of the three dimensions. Only dimension z_1 represents a meaningful distinction of tasks. The representation by PEARL without an RBF layer shows a minor larger spread of the task representations in dimension z_1 than RBF-PEARL (3.676 compared to 3.507). Also, RBF-PEARL has a minor larger spread in dimension z_3 (0.032 compared to 0.003), but both these differences seem negligible.

Both representations show a clear clustering of tasks where tasks with a high reward weight on the visual com-

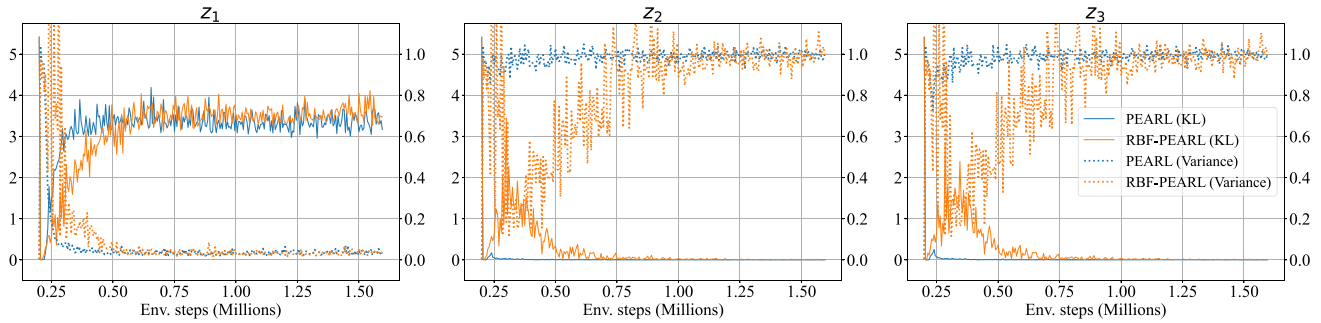


Fig. 13 Kullback-Leibler (KL) divergence and variance of task representation z for the linear gaze control environment: Posterior collapse occurs in two dimensions (z_2, z_3) for both PEARL and RBF-PEARL (note that RBF-PEARL delays the collapse). We report the average KL divergence (solid lines) and variance (dotted lines) of each dimension of the task representation variable z during meta-training. The average is taken over five tasks chosen randomly among the 100 training tasks

ponent (red colored) are on one side in z_1. Representations of the tasks with a high weight on the movement component are clustered at the opposite end (green colored). Tasks with a high weight on the audio component (blue/brown colored) are in the middle. Although the representations (with and without the RBF layer) are inverted to each other, both have this general cluster topology which should therefore not result in a difference in the downstream networks that learn based on them.

We further evaluated whether the learned representation z obtained with the RBF layer has an influence on the performance increase. We trained an actor and critic network with the pre-trained context encoder obtained from PEARL and RBF-PEARL. No significant differences, either in their learning curves or in their final performances, can be observed (Fig. 14). This indicates that differences in the performances of PEARL vs. RBF-PEARL do not result from their differences in the learned representation z.

In summary, the differences between the learned task representation z with and without the RBF layer are minor. They do not explain the increase in the performance of RBF-PEARL.

4.5.2 Influence on the temporal dynamics of learning task representation *z*

We analyzed the temporal dynamics of learning z by looking at how posterior collapse happens on the three dimensions of task representation z. The differences in how posterior collapse occurred between PEARL and RBF-PEARL could explain their performance differences. To measure this, we examine the KL divergence and the variance of the posterior distribution on each of the dimensions of the task representation during the meta-training stage (Fig. 13). Posterior collapse occurs for both methods in two of the three dimensions (z_2, z_3). Nonetheless, RBF-PEARL delays the posterior collapse for 400k steps compared to PEARL. Similarly, for dimension z_1, RBF-PEARL requires more time to learn the final representation as shown by the longer time of the KL loss and variance to reach their asymptotic levels (Fig. 13, left). This delay could explain why RBF-PEARL performs slightly below PEARL for the first 400k steps (Fig. 5, left). Afterward, the KL loss and variance are similar between RBF-PEARL and PEARL.

In conclusion, the RBF layer has a temporal effect on the learning of task representation z by delaying it. This includes a delay in the posterior collapse. This might affect the final performance of the RL algorithm, for example by inducing a greater exploration during learning. Nonetheless, the impact of this effect cannot be clearly defined and we believe it to be of minor consequence to the learning performance.

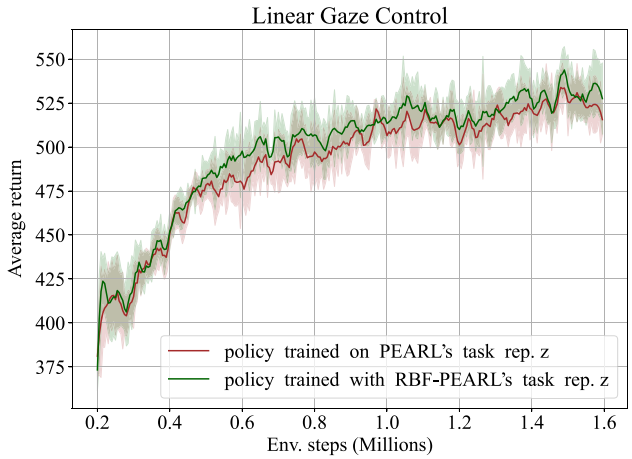


Fig. 14 Performance of policies trained on learned (and frozen) context-encoders from PEARL and RBF-PEARL: Test-task performance during meta-training on the linear gaze-control environment. It shows no significant differences between the two encoders, indicating that the performance difference between PEARL and RBF-PEARL is not due to their differences in the learned task representation z

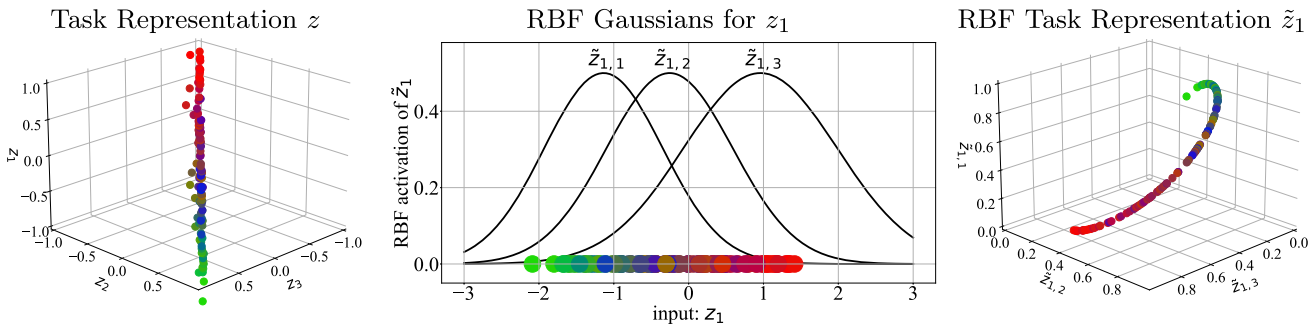


Fig. 15 RBF-PEARL's task representation with the Gaussian associated with dimension z_1 with no posterior collapse: The RBF layer projects task representation $z_1 \in \mathbb{R}$ in a higher dimension $\tilde{z}_1 \in \mathbb{R}^3$, where each RBF Gaussian learns to represent a specific task type. Each colored dot corresponds to the representation of one of the 100 meta-training tasks. The color represents the predominant reward weight component of the reward function, i.e., the task type. Red: largest weight is the visual weight. Blue/brown: high audio weights. Green: high movement weights. (left): Three-dimensional task representation z learned by the task encoder. The tasks are only spread in dimension z_1. The other dimensions (z_2, z_3) have a posterior collapse. (middle): RBF Gaussians associated with the input dimension without posterior collapse (z_1). Each Gaussian specializes in order to represent a different task type. (right): The three-dimensional representation obtained by the RBF layer for input dimension z_1

4.5.3 Influence of the output representation $\tilde{z}$

The final influence that the RBF layer has on the performance of RBF-PEARL is through its output representation $\tilde{z}$, which is given as an input to the downstream policy and value networks instead of z. We visualized this representation for the RBF neurons that encode the non-collapsing dimension z_1 (Fig. 15, right). It lifts the one-dimensional representation z_1 into a three-dimensional space $\tilde{z}_1$. Analyzing the shape of the Gaussians associated with $\tilde{z}_1$ (Fig. 15, middle), each Gaussian is centered around a specific cluster of task representations. The first Gaussian $\tilde{z}_{1,1}$ specializes in tasks with a high weight on the movement component (green colored). Representations of tasks with a high weight on the audio component (blue/brown colored) are centered around the activation region of the second Gaussian $\tilde{z}_{1,2}$. The third Gaussian $\tilde{z}_{1,3}$ specializes in tasks with a high weight on the visual component (red colored). We believe that this effect is the main cause of the RBF layer's performance increase. The objective of the downstream networks is to learn specific policies and value functions for the different tasks, i.e., tasks in which the visual, audio, or movement component is more important. Differentiating between these tasks is difficult from the one-dimensional representation z_1 learned by the standard PEARL. For example, to identify audio tasks which are clustered in the middle of the representation in z_1 (Fig. 2, left-bottom), the downstream networks have to learn a rule that defines this region using two borders: $y > z_1 > x$. In contrast, for RBF representation $\tilde{z}_1$ it is only necessary to identify whether a certain Gaussian has a large activation. In the case of audio tasks, the second Gaussian should be mainly activated: $\tilde{z}_{1,2} > x$. This seems to reduce the complexity of the rules that the downstream networks have to learn to identify tasks, making it easier to learn specific policies and values for them.

In summary, we believe that the main effect that the RBF layer has in improving performances is based on its changed task representation $\tilde{z}$. The representation seems to allow the downstream networks to identify certain tasks more easily and to learn specific outputs for them. Nonetheless, this explanation is only an intuition and should be explored further in future research.

4.6 Open research directions

Meta-RL for social robotics is currently underexplored and several research directions are still open. This section discusses some of these directions.

4.6.1 Partially observable environments

In social robotics, it is often unrealistic to assume that the agent has complete information. Factors such as sensor noise, occlusions, and limited field of view can all contribute to incomplete observation data. Incomplete observations can have a significant impact on the training performance of the agent, which can make it challenging to learn an effective policy for the task at hand [49, 50]. Meta-reinforcement learning is a promising approach to tackle the problem of partially observable environments. By leveraging previous experience and learning how to learn, the agent can quickly adapt to new tasks and environments, even when observation information is incomplete or noisy. Meta-RL has already been studied in the context of adapting to unseen environments with a limited field of view [51] or sim2real scenarios [52], where the agent needs to generalize to new and diverse settings. In our

paper, the gaze control environment can be considered a partially observable environment, as the visual information the robotic head can obtain is limited by the field of view of the camera. However, further research is needed to explore the effectiveness of off-policy meta-RL for partially observable environments in the specific context of social robotics.

4.6.2 Safe reinforcement learning

Safe reinforcement learning can be particularly important in the context of social robotics, where robots are designed to interact with humans in shared environments. Social robots are expected to operate safely and interact with humans in a way that is consistent with social norms and values. While meta-RL in itself cannot guarantee that a robot will operate safely in social environments at test time, it is possible to integrate existing approaches from the literature into meta-RL algorithms. Some common approaches to safe reinforcement learning include adding constraint/regularization methods to the objective function [53], using ensemble networks to capture uncertainties in the environment [54], or using a hierarchical control architecture with a manually designed set of constraints to control the low-level policy [55]. Recent approaches also propose to use meta-RL in environments with non-stationary disturbances to adapt the safety constraints to the disturbances in the environment [56]. Nevertheless, further research is needed to integrate safety constraints into the meta-reinforcement learning framework.

4.6.3 Multi-agent reinforcement learning

In the context of social robotics, multi-agent reinforcement learning (MARL) has been proposed to address situations that involve several agents/robots [57, 58]. In MARL, each agent is represented by a learning algorithm that interacts with the environment and other agents to optimize its performance. The agents can be homogeneous, meaning that they are identical and have the same objective, or heterogeneous, meaning that they have different objectives and learning algorithms. The environment can be either cooperative, where agents work together to achieve a common goal, or competitive, where agents compete with each other to achieve their individual goals.

One of the primary challenges in this setup is to coordinate several continuously learning and differently behaving agents. The changing behavior of one agent can result in a negative outcome for another agent because it is not adapted to the new behavior. Social environments involve human participants, making the problem of generalization even more challenging. Some meta-RL approaches [59, 60] address this by assuming a distribution of agents available for practicing. The resulting policy should then be able to generalize and adapt to environments with agents cooperating in different ways. A similar approach could be used for social robotics where not only are other agents considered for adaptation, but also humans.

5 Conclusion

In this exploratory study, we investigate the use and limitations of variational meta-RL for social robotics. We show that meta-RL successfully learns to adapt quickly (within 200 steps) to a new reward function that describes a desired social behavior in a different environment. Nonetheless, in our task state-of-the-art methods (PEARL) exhibited posterior collapse, which is problematic in meta-RL since the encoder is supposed to accumulate information for better generalization, and collapsed encoding dimensions cannot do so. We started investigating how to mitigate posterior collapse in variational meta-RL by adding an RBF network after each encoded dimension. Based on the result in Table 1, and the analysis of the representation learned by RBF-PEARL, our algorithm improves the performances of meta-RL algorithms for reward design in social robotics. Our RBF layer improves the asymptotic performances of the PEARL algorithm in several different problem domains, all inspired by social robotics tasks. The PEARL algorithm learns a sub-optimal representation of the task. While we do not solve this issue, the RBF-PEARL algorithm mitigates the effect of this sub-optimal representation, providing the actor and critic network with a different representation of the task using the dimensions of the task representation that do not suffer from posterior collapse. The results clearly demonstrate that an RBF layer reduces the effect of posterior collapse, and allows for steeper learning curves and higher asymptotic performance. We believe that such studies pave the way to a better understanding of meta-RL for social robotics, a clearly underinvestigated domain. We hope that our findings will help foster research in this direction.

Funding Partial financial support was received from the ANR MIAI institute (ANR-19-P3IA-0003), H2020 SPRING (#871245), and from the ANR ML3RI (ANR-19-CE33-0008-01). The authors have no competing interests to declare that are relevant to the content of this article.

Code Availability The code used during the current study is available from the corresponding author on reasonable request.

References

1. Fong T, Nourbakhsh I, Dautenhahn K (2003) A survey of socially interactive robots. Robotics and autonomous systems. 42(3–4):143–66
2. Liu H, Liu T, Zhang Z, Sangaiah A, Yang B, Li YA (2022) Asymmetric Relation-Aware Representation Learning for Head Pose Estimation in Industrial Human-Computer Interaction. IEEE Transactions on Industrial Informatics. 18:7107–17

3. Davison DP, Wijnen FM, Charisi V, van der Meij J, Evers V, Reidsma D. Working with a social robot in school: a long-term real-world unsupervised deployment. In: ACM/IEEE international conference on human-robot interaction; 2020. p. 63-72
4. Kubota A, Peterson EI, Rajendren V, Kress-Gazit H, Riek LD. Jessie: Synthesizing social robot behaviors for personalized neurorehabilitation and beyond. In: ACM/IEEE international conference on human-robot interaction; 2020. p. 121-30
5. Colledanchise M, Ögren P. Behavior trees in robotics and AI: An introduction. CRC Press; 2018
6. Sutton RS, Barto AG. Reinforcement learning: An introduction. MIT press; 2018
7. Akalin N, Loutfi A (2021) Reinforcement learning approaches in social robotics. Sensors. 21(4):1292
8. Beck J, Vuorio R, Liu EZ, Xiong Z, Zintgraf L, Finn C et al (2023) A Survey of Meta-Reinforcement Learning. arXiv preprint arXiv:2301.08028
9. Chen YF, Liu M, Everett M, How JP. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In: IEEE International Conference on Robotics and Automation; 2017. p. 285-92
10. Zhou Z, Zhu P, Zeng Z, Xiao J, Lu H, Zhou Z. Robot navigation in a crowd by integrating deep reinforcement learning and online planning. Applied Intelligence. 2022:1-17
11. Li C, Castellano G, Gao Y. Efficient Learning of Socially Aware Robot Approaching Behavior Toward Groups via Meta-Reinforcement Learning. In: IEEE/RSJ International Conference on Intelligent Robots and Systems; 2020. p. 12156-9
12. Kingma DP, Welling M. Auto-encoding variational bayes. In: International Conference on Learning Representatnions; 2014.
13. Rakelly K, Zhou A, Finn C, Levine S, Quillen D. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In: International conference on machine learning; 2019. p. 5331-40
14. Liu T, Wang J, Yang B, Wang X (2021) NGDNet: Nonuniform Gaussian-label distribution learning for infrared head pose estimation and on-task behavior understanding in the classroom. Neurocomputing. 436:210–20
15. Girin L, Leglaive S, Bie X, Diard J, Hueber T, Alameda-Pineda X (2021) Dynamical Variational Autoencoders: A Comprehensive Review. Foundations and Trends in Machine Learning. 15(1–2):1–175
16. Broomhead DS, Lowe D. Radial basis functions, multi-variable functional interpolation and adaptive networks. Royal Signals and Radar Establishment Malvern; 1988
17. Park J, Sandberg IW (1991) Universal approximation using radial-basis-function networks. Neural computation. 3(2):246–57
18. Vidnerová P, Neruda R. Deep networks with rbf layers to prevent adversarial examples. In: Artificial Intelligence and Soft Computing. Springer; 2018. p. 257-66
19. Abpeykar S, Ghatee M, Zare H (2019) Ensemble decision forest of RBF networks via hybrid feature clustering approach for high-dimensional data classification. Computational Statistics & Data Analysis. 131:12–36
20. Asadi K, Parikh N, Parr RE, Konidaris GD, Littman ML. Deep radial-basis value functions for continuous control. In: AAAI Conference on Artificial Intelligence. vol. 35; 2021. p. 6696-704
21. Möller R, Furnari A, Battiato S, Härmä A, Farinella GM (2021) A survey on human-aware robot navigation. Robotics and Autonomous Systems. 145:103837
22. Lathuilière S, Massé B, Mesejo P, Horaud R (2019) Neural network based reinforcement learning for audio-visual gaze control in human-robot interaction. Pattern Recognition Letters. 118:61–71
23. Breazeal C, Dautenhahn K, Kanda T. Social robotics. Springer handbook of robotics. 2016:1935-72
24. Sheridan TB (2020) A review of recent research in social robotics. Current opinion in psychology. 36:7–12
25. Henschel A, Laban G, Cross ES (2021) What makes a robot social? a review of social robots from science fiction to a home or hospital near you. Current Robotics Reports. 2:9–19
26. Ahmad MI, Mubin O, Orlando J (2017) A systematic review of adaptivity in human-robot interaction. Multimodal Technologies and Interaction. 1(3):14
27. Martins GS, Santos L, Dias J (2019) User-adaptive interaction in social robots: A survey focusing on non-physical interaction. International Journal of Social Robotics. 11:185–205
28. Nocentini O, Fiorini L, Acerbi G, Sorrentino A, Mancioppi G, Cavallo F (2019) A survey of behavioral models for social robots. Robotics. 8(3):54
29. Patompak P, Jeong S, Nilkhamhang I, Chong NY (2020) Learning proxemics for personalized human-robot social interaction. International Journal of Social Robotics. 12:267–80
30. Choi J, Dance C, Kim Je, Park Ks, Han J, Seo J, et al. Fast adaptation of deep reinforcement learning-based navigation skills to human preference. In: IEEE International Conference on Robotics and Automation; 2020. p. 3363-70
31. Oh J, Hessel M, Czarnecki WM, Xu Z, van Hasselt HP, Singh S et al (2020) Discovering reinforcement learning algorithms. Advances in Neural Information Processing Systems. 33:1060–70
32. Kirsch L, van Steenkiste S, Schmidhuber J. Improving generalization in meta reinforcement learning using learned objectives. In: International Conference on Learning Representations; 2020.
33. Finn C, Abbeel P, Levine S. Model-agnostic meta-learning for fast adaptation of deep networks. In: International Conference on Machine Learning; 2017. p. 1126-35
34. Duan Y, Schulman J, Chen X, Bartlett PL, Sutskever I, Abbeel P (2016) Fast reinforcement learning via slow reinforcement learning. arXiv preprint arXiv:1611.02779
35. Zhao Z, Nagabandi A, Rakelly K, Finn C, Levine S. MELD: Meta-Reinforcement Learning from Images via Latent State Models. In: Conference on Robot Learning; 2021. p. 1246-61
36. Dai B, Wang Z, Wipf D. The usual suspects? Reassessing blame for VAE posterior collapse. In: International Conference on Machine Learning; 2020. p. 2313-22
37. Zhao S, Song J, Ermon S (2017) Infovae: Information maximizing variational autoencoders. arXiv preprint arXiv:1706.02262
38. Razavi A, van den Oord A, Poole B, Vinyals O. Preventing Posterior Collapse with delta-VAEs. In: International Conference on Learning Representations; 2019
39. Sønderby CK, Raiko T, Maaløe L, Sønderby SK, Winther O. Ladder variational autoencoders. In: Advances in neural information processing systems; 2016.
40. Huang CW, Tan S, Lacoste A, Courville AC. Improving explorability in variational inference with annealed variational objectives. In: Advances in neural information processing systems; 2018.
41. Reinke C, Etcheverry M, Oudeyer PY. Intrinsically motivated exploration for automated discovery of patterns in morphogenetic systems. In: International Conference on Learning Representations; 2020.
42. He J, Spokoyny D, Neubig G, Berg-Kirkpatrick T. Lagging Inference Networks and Posterior Collapse in Variational Autoencoders. In: International Conference on Learning Representations; 2019.
43. Haarnoja T, Zhou A, Abbeel P, Levine S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International conference on machine learning; 2018.
44. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D et al (2013) Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602
45. GitHub (2019) Garage: A toolkit for reproducible reinforcement learning research. https://github.com/rlworkgroup/garage

46. Pedica C, Vilhjálmsson H. Social perception and steering for online avatars. In: International Workshop on Intelligent Virtual Agents. Springer; 2008. p. 104-16
47. Satake S, Kanda T, Glas DF, Imai M, Ishiguro H, Hagita N (2012) A robot that approaches pedestrians. IEEE Transactions on Robotics. 29(2):508–24
48. Reinke C, Alameda-Pineda X (2023) Successor feature representations. Transactions on Machine Learning Research
49. Quintero-Pena C, Chamzas C, Sun Z, Unhelkar V, Kavraki LE. Human-Guided Motion Planning in Partially Observable Environments. In: IEEE International Conference on Robotics and Automation; 2022. p. 7226-32
50. Rosano M, Furnari A, Gulino L, Farinella GM. On embodied visual navigation in real environments through habitat. In: IAPR International Conference on Pattern Recognition; 2021. p. 9740-7
51. Wortsman M, Ehsani K, Rastegari M, Farhadi A, Mottaghi R. Learning to learn how to learn: Self-adaptive visual navigation using meta-learning. In: IEEE/CVF conference on computer vision and pattern recognition; 2019. p. 6750-9
52. Arndt K, Hazara M, Ghadirzadeh A, Kyrki V. Meta reinforcement learning for sim-to-real domain adaptation. In: IEEE international conference on robotics and automation; 2020. p. 2725-31
53. Yang Q, Simão TD, Tindemans SH, Spaan MT. WCSAC: Worst-case soft actor critic for safety-constrained reinforcement learning. In: AAAI Conference on Artificial Intelligence; 2021. p. 10639-46
54. Lütjens B, Everett M, How JP. Safe reinforcement learning with model uncertainty estimates. In: International Conference on Robotics and Automation; 2019. p. 8662-8
55. Xiong Z, Agarwal I, Jagannathan S. HiSaRL: A Hierarchical Framework for Safe Reinforcement Learning. In: SafeAI@ AAAI; 2022.
56. Chen B, Liu Z, Zhu J, Xu M, Ding W, Li L, et al. Context-aware safe reinforcement learning for non-stationary environments. In: IEEE International Conference on Robotics and Automation; 2021. p. 10689-95
57. Everett M, Chen YF, How JP. Motion planning among dynamic, decision-making agents with deep reinforcement learning. In: IEEE/RSJ International Conference on Intelligent Robots and Systems; 2018. p. 3052-9
58. Semnani SH, Liu H, Everett M, De Ruiter A, How JP (2020) Multi-agent motion planning for dense and dynamic environments via deep reinforcement learning. IEEE Robotics and Automation Letters. 5(2):3221–6
59. Charakorn R, Manoonpong P, Dilokthanakul N. Learning to Cooperate with Unseen Agents Through Meta-Reinforcement Learning. In: International Conference on Autonomous Agents and MultiAgent Systems; 2021. p. 1478-9
60. He JZY, Erickson Z, Brown DS, Raghunathan A, Dragan A. Learning Representations that Enable Generalization in Assistive Tasks. In: Conference on Robot Learning; 2023. p. 2105-14

Anand Ballou received his M.Sc. degree in computer science and applied mathematics from Grenoble INP - Ensimag, France in 2019. He is currently working as a Ph.D. student in the RobotLearn team at the Inria center of the University Grenoble Alpes, France, under the supervision of Dr. Chris Reinke and Dr. Xavier Alameda-Pineda. His research areas include deep learning, reinforcement learning, preference-based learning, meta-learning, robotics, and human-robot interaction.

Xavier Alameda-Pineda is a (tenured) Research Scientist at Inria and the Leader of the Robot Learn Team. He obtained the M.Sc. (equivalent) in Mathematics in 2008, in Telecommunications in 2009 from BarcelonaTech and in Computer Science in 2010 from Université Grenoble-Alpes (UGA). He then worked towards his Ph.D. in Mathematics and Computer Science, and obtained it in 2013, from UGA. After a two-year post-doc period at the Multimodal Human Understanding Group, at the University of Trento, he was appointed to his current position. Xavier is an active member of SIGMM, a senior member of IEEE, and a member of ELLIS. He is the Coordinator of the H2020 Project SPRING: Socially Pertinent Robots in Gerontological Healthcare and is co-leading the "Audio-visual machine perception and interaction for companion robots" chair of the Multidisciplinary Institute of Artificial Intelligence. Xavier's research interests are at the crossroads of machine learning, computer vision, and audio processing for scene and behavior analysis and human-robot interaction.

Chris Reinke is currently a post-doctoral researcher in the Robot Learn team at Inria Grenoble (France). He is leading the Learning Robot Behavior work package of the European H2020 SPRING project (Socially Pertinent Robots in Gerontological Healthcare) which explores deep reinforcement learning methods and their application to social robotics. He received his Bachelor's (2009) and Master's degree (2012) in Cognitive Science from the University of Osnabrück (Germany). He pursued his Ph.D. degree in the Neural Computation Unit of Prof. Dr. Kenji Doya at the Okinawa Institute of Science and Technology (OIST) and obtained it in 2018. Before starting his current position in 2020 he was a postdoctoral researcher in the Flowers team of Pierre-Yves Oudeyer located in Bordeaux (France). His research interests are artificial intelligence, machine learning, and cognitive science with a main focus on adaptive learning mechanisms.