Honda Research Institute EU
Honda Research Institute Europe GmbH
https://www.honda-ri.de/

Reasoning about Uncertain Parameters and Agent Behaviors through Encoded Experiences and Belief Planning

Akinobu Hayashi, Dirk Ruiken, Christian Goerick, Tadaaki Hasegawa

2019

Preprint:

This is an accepted article published in Artificial Intelligence, Special Issue on Autonomous Agents Modelling Other Agents. The final authenticated version is available online at: https://doi.org/[DOI not available]

Artificial Intelligence 280 (2020) 103228

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Reasoning about uncertain parameters and agent behaviors through encoded experiences and belief planning

Akinobu Hayashi [a,*], Dirk Ruiken [a], Tadaaki Hasegawa [b], Christian Goerick [a]

[a] Honda Research Institute Europe GmbH, Offenbach/Main, Germany
[b] Honda R&D Co., Ltd. Center X, Tokyo, Japan

ARTICLE INFO

Article history:
Received 18 February 2019
Received in revised form 9 December 2019
Accepted 18 December 2019
Available online 27 December 2019

Keywords:
Planning under uncertainty
Partial observability
Cooperative multi-agent system
Parameter estimation
Object manipulation
Model adaptation

ABSTRACT

Robots are expected to handle increasingly complex tasks. Such tasks often include interaction with objects or collaboration with other agents. One of the key challenges for reasoning in such situations is the lack of accurate models that hinders the effectiveness of planners. We present a system for online model adaptation that continuously validates and improves models while solving tasks with a belief space planner. We employ the well known online belief planner POMCP. Particles are used to represent hypotheses about the current state and about models of the world. They are sufficient to configure a simulator to provide transition and observation models. We propose an enhanced particle reinvigoration process that leverages prior experiences encoded in a recurrent neural network (RNN). The network is trained through interaction with a large variety of object and agent parametrizations. The RNN is combined with a mixture density network (MDN) to process the current history of observations in order to propose suitable particles and models parametrizations. The proposed method also ensures that newly generated particles are consistent with the current history. These enhancements to the particle reinvigoration process help alleviate problems arising from poor sampling quality in large state spaces and enable handling of dynamics with discontinuities. The proposed approach can be applied to a variety of domains depending on what uncertainty the decision maker needs to reason about. We evaluate the approach with experiments in several domains and compare against other state-of-the-art methods. Experiments are done in a collaborative multi-agent and a single agent object manipulation domain. The experiments are performed both in simulation and on a real robot. The framework handles reasoning with uncertain agent behaviors and with unknown object and environment parametrizations well. The results show good performance and indicate that the proposed approach can improve existing state-of-the-art methods.

© 2019 Elsevier B.V. All rights reserved.

1. Introduction

Robots are increasingly required to complete tasks in the real-world by themselves or to achieve effective collaboration as a partner to a human or another robot. To act appropriately, robots need some understanding of situations or contexts. They can select useful actions based on prior experience, e.g. simply follow a learned policy, or use their knowledge on how

* Corresponding author.
E-mail address: Akinobu_Hayashi@n.f.rd.honda.co.jp (A. Hayashi).

https://doi.org/10.1016/j.artint.2019.103228
0004-3702/© 2019 Elsevier B.V. All rights reserved.

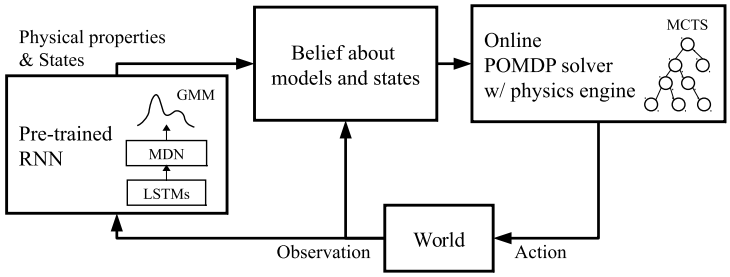


Fig. 1. System overview: The state of the world is represented as a belief over model parametrizations and states. An online POMDP solver uses Monte-Carlo tree search (MCTS) to select the best next action to take. Particles contain both state and model information that can be used to configure a physics engine to provide black box transition and observation functions. The belief is updated based on the taken action and new observations. For particle reinvigoration the history of observations is passed to a pre-trained RNN that feeds a mixture density networks (MDN). The output Gaussian mixture models (GMM) provide potential model parametrizations and states that are added as new particles to the belief. This process repeats until the task is completed. The new models provided by the Long Short-Term Memory (LSTM) networks and MDN progressively adapt and improve the model parameters to better match the actual situation.

they can impact the world, e.g. from prediction models, to reason about viable strategies. Planners can effectively make use of such prediction models to find good solutions. However, to be successful, they must overcome the lack of sufficiently expressive or accurate prediction models for real-world objects, environments, or other agents behaviors. Usually prediction models mismatch the real situation. Variations in an object, an environment, or an agent behavior can be represented as different types and parameters in models, though there will usually be some mismatches between model and reality. Often the models are also unable to capture the discontinuities in the dynamics that can arise in object manipulation. In object manipulation, contact-rich tasks (e.g. peg insertion tasks) where the true system dynamics are not only nonlinear but even discontinuous are often required to be completed. Besides, in partially observed environments, unforeseen contact discontinuities due to the hidden surrounding objects can happen even in a simple task such as object pushing. All of this can lead the planners that rely on these models to poor performance as the actual plan execution might deviate a lot from the anticipated execution. In addition, many details about a situation are unknown due to partial observability and uncertainty in observation and action outcomes. Such tasks can be represented as partially observable Markov decision processes (POMDPs) [37,21]. Belief space planners have been shown to handle both partial observability and uncertainty of POMDPs, but they also rely on appropriate models and typically scale very poorly in computation time with the complexity of these models (and the resulting state, action, and observation spaces). Sampling-based POMDP planners such as POMCP [36] and DESPOT [38] have been shown to alleviate this problem as they can handle much larger state, action, and observation spaces, but they still struggle with increasing domain complexity that is encountered in complex robotic applications as they rely on good samples to be efficient. Our approach aims to alleviate this problem by leveraging past experiences to generate better and more meaningful samples for such planners.

The goal of this work is to develop a system for online model adaptation that exploits both current observations *as well as past experiences* in order to improve an agent's belief over models of the situation and thus improves task performance. Specifically, we want to evaluate if we can use knowledge from prior experiences to improve the particle invigoration process. A success here would alleviate problems arising from poor sampling quality and also enable larger jumps in suggested particles. As a result the quality of the adapted models and also the task success rate would increase. This could also prevent or reduce particle deprivation while planning in domains with dynamics with discontinuities. Specifically, in this work, our contributions are a neural network based particle reinvigoration process that can leverage past experiences encoded in the network and a process for ensuring that all generated particles are consistent will all previous observations and not just the current one.

Our framework uses the well-known online POMDP solver POMCP [36], but we adapt the belief and the particle reinvigoration process and use a simulator to obtain transition and observation functions. The planning portion of POMCP that is used to choose (good) actions given the current belief and a task is untouched. The belief is managed by a particle filter. Each particle represents an hypothesis of the current situation and contains two types of (usually uncertain) information: 1) the state: dynamic aspects such as the position of a robot or an object; 2) a model: static aspects such as fixed environment properties or agents' behavior types and capabilities. The state and model can be used to configure a simulator which then serves as black-box transition and observation function. As initial models likely do not include a model that matches the real situation well enough, after each action a particle reinvigoration is performed. Our new particle reinvigoration method uses a recurrent neural network (RNN) [33] trained on prior experiences in combination with a mixture density network (MDN) [6] to continuously suggest suitable models or model parametrizations based on real observations. These models are then added in new particles to be added to the current belief after they are checked for consistency with the history of observations. Thus, after each action, the RNN uses the history of observations to provide refined models. The models are progressively adapted and validated to the actual situation while non-matching models lose support and impact on the planner. Fig. 1 illustrates an overview of our approach.

The approach is evaluated in two domains that require collaboration with other agents with unknown behavior and manipulation of unknown objects and environments respectively:

1. The level-based foraging domain [1,2] is a test domain for *ad hoc teamwork* [39]. Here, a single agent under our control needs to collect items collaboratively with other unknown agents. In order to succeed, suitable actions have to be selected while taking the other agents into account. Their behavior is based on their fixed (behavior) type and sensing capabilities (parameters) as well as an internal state that depends on their history of observations. This dependence on the internal state makes the estimation of types and the parameters difficult as they are hardly distinguishable from short sequences of observations. In addition to the standard domain we demonstrate the effectiveness of our approach in an extension of this domain by introducing new partial observability in the perception of the controlled agent.
2. An object manipulation domain is used to evaluate the capability of a robot to solve manipulation tasks while adapting models to match unknown objects and environments. Apart from action uncertainty, the agent has to consider partial observability of several aspects: (i) objects of unknown shape type and dimensions; (ii) objects of unknown friction and mass; (iii) the presence and properties of an obstacle in the environment; (iv) lack of knowledge as to whether some objects are movable or fixed. This domain also exhibits dynamics with discontinuities that generally pose difficulties to model adaptation methods. Experiments are done both in simulations and on a real robot. To the best of our knowledge, there are no algorithms for handling the planning of robot manipulation actions well while considering partial observability of these aspects.

For each domain we compare the performance with and without the improved reinvigoration process as well as a state-of-the-art method.

Our approach can handle estimation and adaptation of models for both hypothetical behaviors for other agents in a cooperative tasks and for object manipulation in presence of dynamics with discontinuities. Note that, while models are very general and can contain a mix of both discrete or continuous parameters, the simulator needs to be able to interpret these parameters. Therefore, it is possible that parameter adaptation is limited to choose from a predefined set or to obey some logical constraints. For example, in the multi-agent domain the behavior type of another agent can only be chosen from the finite set of pre-determined types. Similarly, for a given type the corresponding continuous model parameters can be adjusted freely but only within bounds that are logical for that type, e.g. there cannot be a negative viewing distance or a sphere with a negative radius.

The remainder of this paper is organized as follows: Section 2 discusses related work with respect to model adaptation and POMDP planning. Our approach is introduced in Section 3. Experiments and results for multi-agent and object manipulation domains are presented in Section 4 and 5 respectively. Finally, Section 6 provides a conclusion and a discussion of future work.

2. Related work

There are many different approaches of handling uncertainty in the different fields. Those approaches deal with the problem of planning appropriate actions in environments where i) teammates or manipulation targets are unknown *a priori*, ii) are partially observable, and iii) action outcomes are uncertain.

2.1. Adaptive control and policy adaptation

Adaptive control methods are used to adapt to a system with parameters which are initially unknown or time-varying [24,4]. One of the simple but powerful approaches to perform adaptation is to use online stochastic gradient descent (SGD) on the model parameters. In recent years, policy-based and value-based reinforcement learning using deep neural network (DNN) have significantly evolved [35,27]. However, such modern parametric models are not reasonable to train online with SGD based approaches since they usually require medium-sized data and multiple gradient steps to reach to better parameters. Also, learned policies or value functions are usually difficult to adapt to different domains or new settings [29]. While meta-learning such as model-agnostic meta-learning (MAML) [12], that learns a prior initialization for the parameters to enable rapid online adaptation, has recently reduced the necessary numbers of gradient steps, medium-sized data is still needed. Because of the high data requirement and the difficulties to adapt to new settings, we focus on model-based or type-base approaches.

2.2. Model learning and online adaptation

Model-based approaches aim at learning dynamics from experiences. These approaches can be divided into two groups: they either try to learn a single global model that should be accurate in most of the state space or they try to learn local models that are fitted only around current experience or policy. To learn a single global model, usually a general function approximator such as a Gaussian process (GP) [10] or a deep neural network (DNN) [31,13] is used. A key challenge with these approaches is the difficulty of learning a global model that is accurate for the entire state space. However, GPs scale poorly with increasing amount of training data. On the other hand, DNNs require a huge amount of data to train a

good global model. Our approach relaxes the need to learn a single accurate global model by allowing multiple uncertain local models to be adapted to different scenarios based on recent observations. Since the planned actions take account of uncertainties over the multiple hypothetical models that are continuously updated online based on recent observations, the approach enables compensation for more substantial discontinuities in the dynamics. Our approach can be viewed as learning local models online by combining a small amount of recent experiences and a learned global prior.

The local model approach only considers dynamics to explain experiences from a specific state to a target state. In the case of having multiple types of parametric models, it is necessary to fit all models to the obtained experience and select the best one by using the information criteria. Sturm et al. compute the maximum likelihood parameters of each model, and select the best one by using an information criterion such as the Bayesian information criterion [40]. Levine and Abbeel iteratively apply time-varying linear-Gaussian dynamics for fitting a local linear quadratic regulator to update for guided policy search under unknown dynamics [25]. Fu et al. use a neural network that encode prior knowledge to predict model parametrizations and can cope with small changes [14]. However, these local model methods often have a gap between training and testing (or real world). In this paper, we deal with this gap by continuously validating and adapting the model parameters while planning and executing actions to complete the task.

Because local model approaches estimate local dynamics, it is also difficult to adapt to dynamics with discontinuous transitions or sudden jumps. Chebotar et al. combined model-based and model-free approaches and improved handling of dynamics with discontinuous transitions [8]. Instead of preparing many local models, there is also a nonparametric approach to infer the model parameters by using a physics simulation as a hypothesis space [34]. From the viewpoint of expressive power of the model, there is applicability to dynamics with discontinuities, but the search space is large and it is difficult to scale up, which is also a problem when assuming an accurate simulator. Our approach uses a NN to learn a global model for the parameters of a physics engine. This global model does not need to be very accurate as it will only provide priors for the parameters which will then be refined at runtime. The physics engine can then handle dynamics with discontinuities.

2.3. POMDP planning

POMDPs offer a general framework for decision making under imperfect information [37,21]. Although POMDP solvers scale very poorly with a growing number of states, observations, and actions and are computationally intractable in the worst case [28], various types of approximation algorithms have been leveraged. There are two main approaches to POMDP planning: online and offline solvers. Global solvers are used for offline policy computation and online search. Global belief-based approaches that evaluate the wide range of possibilities over the assumed world offline scale poorly with increasing problem size. Progress to improve their scalability has slowed down after the work by Kurniawati et al. [23]. In contrast, local online planners have still scaled-up [30,36,38]. During the online search, most algorithms explicitly represents the belief as a probability distribution over the state space. This, however, limits their scalability for large state spaces, because a single belief update can take time quadratic in the number of states. In contrast, POMCP [36] and DESPOT [38] algorithms represent the belief as a set of particles, and do not perform a full belief update during the online search. As both POMCP and DESPOT rely on good samples to be efficient, both again struggle with increasing number of domains or more complex robotic applications like manipulation tasks. In the proposed framework, we use POMCP, but adapt the reinvigoration method.

2.4. Modeling other agents

One of the methods of modeling other agents is called *type-based reasoning* as discussed in a recent survey [3]. Type-based reasoning methods assume that an agent is able to reuse previously learned knowledge (e.g. a finite set of potential types for other agents) instead of learning new models from scratch. Our approach can perform type-based reasoning as it progressively chooses which type fits best to the current history as well as adjusting parameters accordingly planning actions to complete the task collaboratively with other agents. There are several works which use artificial neural networks instead of Bayes' law to estimate the relative likelihood of agent types from observed actions. Lockett et al. proposed a method to learn mixture weights over a pre-specified set of types to identify an opponent's type [26]. He et al. proposed the deep reinforcement opponent network, that predicts Q-values and infers an opponent strategy jointly [19]. Since we use a Mixture Density Networks (MDN) [6] for the network outputs, it can learn a multi-modal probability density over types and parameters. Moreover, we use the outputs of the network as hypotheses that can be merged to a belief state which is used in a POMDP planner. We blend the Bayesian filtering and the encoded experience based hypothesis generation. One of the approaches to collaborate with unknown teammates without prior coordination is online adaptation of the policy by using knowledge from past interactions. Wu et al. [41] proposed the online approach to bias the agent's action selection according to the teammates' past actions. Barrett and Stone [5] used reinforcement learning to learn policies of how to cooperate with previous teammates and then reuse this knowledge to cooperate with new teammates. Our approach does online planning of actions while adapting the belief over the model types and parameters with checking the consistency of action-observation history. Another approach is to explicitly model and reason about other agents' beliefs such as the work on Interactive POMDPs (I-POMDPs) [15] and Interactive Dynamic Influence Diagram (I-DID) [11]. However, modeling other agents' beliefs struggles large scale problems since it greatly expands the planning space. Our approach uses standard

POMDPs, and is designed to enhance the scalability to larger problems by using the combination of online POMDP solver and the pre-trained RNN.

2.5. Neural network architecture

Our architecture uses a combination of Long Short-Term Memory (LSTM) recurrent neural networks [20] and Mixture Density Networks (MDN) [6] to predict the probability density over the multi-modal output value space. For regression problems, NNs usually output a single value, e.g. $\mu(x)$, and the parameters are optimized to minimize the mean squared error on the training data set. MDN was proposed to predict multi-modal data [6]. The network outputs the parameters of a mixture distribution. Gaussian mixture model (GMM) with the predicted mean, variance, and mixing coefficient, minimizing the negative log-likelihood. LSTM recurrent neural networks are suitable for sequential data [20]. The performance of combination of MDN with LSTM has been demonstrated with an example to generate artificial handwriting [17] and learning robot trajectories [32].

3. Approach

We propose a method for online model adaptation and planning. The approach handles partial observability and uncertainty despite (initially) imprecise models and even tasks to be done together with agents with uncertain behavior parametrization, or object manipulation tasks with discontinuities in the dynamics. The main components are a neural network based particle reinvigoration process to leverage prior experiences to produce better particles and a method for ensuring that only particles that are consistent with the whole history of observations are added.

First, we introduce how a task can be represented as a POMDP, how we can handle uncertain types and parameters by integrating them into transition and observation functions for use in a behavior simulator or a physics engine (Section 3.1). Second, we describe how an established planner is used for solving tasks in such domains (Section 3.2). After that, we discuss how models can be updated while keeping consistency with the history of actions and observations (Section 3.3). Then, we present the recurrent neural network that is used to continuously suggest suitable models or model parameterizations from the history of actions and observations (Section 3.3.2). Finally, we describe how the individual parts are integrated to perform model adaptation while completing the task (Section 3.4).

3.1. Belief representation

POMDPs are well suited for formulating planning problems in the presence of uncertainty and partial observability. A POMDP is defined as a tuple:

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{Z}, \mathcal{R}, \gamma, b_0 \rangle, \tag{1}$$

where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{O}$ is the set of observations, $\mathcal{P} = \Pr(s_{t+1} \mid s_t, a_t)$ is the stochastic transition function, $\mathcal{Z} = \Pr(o_{t+1} \mid s_{t+1}, a_t)$ is the stochastic observation function, $\mathcal{R}(s_t, a_t)$ is the reward function, $\gamma \in [0, 1]$ is the discount factor, and b_0 is the belief about the initial state. $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}$, $o_t \in \mathcal{O}$ are the state, action, and observation at time step t, respectively. At each step, the agent receives a reward $r_{t+1} = \mathcal{R}(s_t, a_t)$. The goal is to compute the optimal policy $\pi^* : b \to a$ that maximizes the expected discounted total reward $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)\pi(b_t, a) \mid b_0]$ from a certain belief state $b_0 \in \mathcal{B}$.

The ability to plan in such a domain depends on the transition and observation models. A general model for stochastic dynamics and observations can be defined as:

$$s_{t+1} \sim f(s_t, a_t), \tag{2}$$

$$o_{t+1} \sim g(s_{t+1}, a_t). \tag{3}$$

Here, usually both f and g are fixed functions. For smaller problems these functions can easily be defined *a priori*. However, for more complex situations, designing or learning a single global dynamics model that accurately covers the state space is challenging. Often simulations are used as a black box to calculate other agent's behaviors or results of physical interactions at runtime though these simulations still depend on accurate parameters. In this paper, we use a such a simulator to provide the transition and observation functions f and g and combine it with online model adaptation to adjust imperfect models from observed experiences. For a given state s_t and action a_t the simulator can predict next state s_{t+1} and observation o_{t+1}, but it needs to be configured with a model of the environment or other agents. The quality of the prediction depends on how well the model(s) match the real situation. A model is defined as a set parameters (Θ) that represent all aspects that are needed to configure the simulator apart from state s and action a. For example, these could range from agent behaviors to physical properties of world models. For the remainder of the paper we use Θ to refer both to the model and the parameters that define it. The updated transition and observation functions depend on Θ:

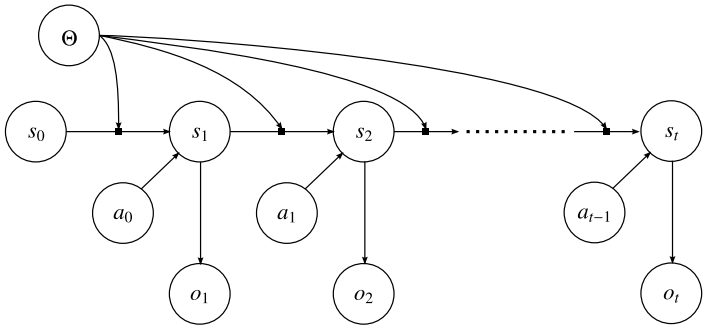


Fig. 2. Graphical model representing the online model adaptation/learning problem.

$$s_{t+1} \sim f_\Theta(s_t, a_t \mid \Theta), \tag{4}$$

$$o_{t+1} \sim g_\Theta(s_{t+1}, a_t \mid \Theta). \tag{5}$$

Fig. 2 depicts the graphical model of our definition.

We define Θ as a set of time independent parameters, containing multiple groups of both *discrete* variables $\Delta = \{d_1, \ldots, d_n\}$ and *continuous* variables $\Phi = \{p_1, \ldots, p_m\}$:

$$\Theta := \{\{\Delta, \Phi\}_1, \ldots, \{\Delta, \Phi\}_k\}. \tag{6}$$

For example, in the level-based foraging domain (Section 4) an agent reasons about the behaviors of other agents that are specified with both a discrete type $\{d_{type}\}$ (see Section 4.1 for a set of known possible behavior types) and corresponding continuous parameters $\{p_1, p_2, p_3\}$ for the behavior type. With our definition, $\theta_{agents} \in \Theta$ for k agents can be represented as:

$$\theta_{agents} = \left\{\left\{d_{type}, p_1, p_2, p_3\right\}_1, \ldots, \left\{d_{type}, p_1, p_2, p_3\right\}_k\right\}. \tag{7}$$

Similarly, in an object manipulation domain, $\theta_{objects} \in \Theta$ for k objects can consist of parameters for object dynamics such as mass m, the location of the center of mass $\boldsymbol{x}_{com}$, and friction coefficients μ_c, together with geometric information such as shape types d_{type} and sizes $\boldsymbol{p}_{size}$ (see Section 5.2 for a set of known possible shape types).

$$\theta_{objects} = \left\{\left\{d_{type}, m, \boldsymbol{x}_{com}, \mu_c, \boldsymbol{p}_{size}\right\}_1, \ldots, \left\{d_{type}, m, \boldsymbol{x}_{com}, \mu_c, \boldsymbol{p}_{size}\right\}_k\right\}. \tag{8}$$

Some of these parameters, such as mass, are not directly observable but still influence the transition dynamics and can thus be estimated based on the difference between expected and actual observations.

In a POMDP, the current state s_t is usually not known with certainty. Instead, the belief over the possible states at time t is used: $b_t(s)$. This belief is used to summarize the probability of the world state based on the histories of actions and observations: $h_t = \{a_0, o_1, \ldots, a_{t-1}, o_t\}$. With the introduction of parameters Θ, we extend the belief to include both the state s and physical parameters Θ, and it can be described by a conditional joint distribution as

$$b_t(s, \Theta) = \Pr(s_t, \Theta \mid h_t). \tag{9}$$

3.2. POMDP planning

We use POMCP [36] as an online planner. It scales up to complex problems by representing the current belief as a set of particles, the use of Monte-Carlo Tree Search (MCTS) for the forward search from the current belief state, and the use of Upper Confidence bounds applied to Trees [22]. The sampling based online POMDP solvers which use a black box simulator [36,38] are suitable for the problem of dynamics fluctuation (e.g. sudden jump in dynamics) since they can easily handle multiple transition functions and observation functions even within a single forward search. We use a physics engine as a black box simulator and generative model to provide transition and observations functions. We use POMCP with root parallelization [7], and domain knowledge by defining a set of *preferred actions* that the rollout policy selects uniformly from as in [36].

In standard POMCP, belief $b(s)$ is represented by a set of particles of state s. By sampling an initial state from the belief, the simulation step for planning using a generative model $\mathcal{G}$ is:

$$(s_{t+1}, o_{t+1}, r_{t+1}) \sim \mathcal{G}(s_t, a_t). \tag{10}$$

To choose the next action, it builds a search tree $T(h_t)$ from the current history or belief by performing many simulations. The start state for each simulation is sampled from the particles of belief $b(s)$. Once the search is completed, the action

with the best value a_t is executed and real observations o_{t+1} are received. The belief is updated based on new action and observation. Now, the corresponding subtree $T(h_{t+1}) = T(h_t a_t o_{t+1})$ for action a_t and observation o_{t+1} becomes the new root of the search tree and all other branches of the previous root can be pruned as they are now impossible to reach. The remaining search tree can be reused for the next iteration. This procedure is repeated until the goal is reached.

As the belief in our framework is changed from $b_t(s)$ to $b_t(s, \Theta)$ to include parameters Θ (see Sec. 3.1), the standard POMCP approach has to be adjusted minimally. In addition to state s each particle also contains the parameters Θ that is used to configure the simulator and with that the generative model $\mathcal{G}$. Equation (10) is updated to:

$$(s_{t+1}, o_{t+1}, r_{t+1}) \sim \mathcal{G}(s_t, a_t; \Theta). \tag{11}$$

It has to be noted that with complex and continuous observations the probability of finding an existing subtree that precisely matches the real action and observation in the search tree is very low. Therefore, in our framework the pruning steps is skipped and a new search tree $T(h_t a_t o_{t+1})$ is built from scratch based on actual action a_t and observation o_{t+1}.

3.3. Belief update and particle reinvigoration

Belief is usually updated by using Bayesian filtering to incorporate the effects of the taken action a_t and the new observation o_{t+1}:

$$\hat{b}_{t+1}(s, \Theta) = \sum_{s_t \in \mathcal{S}} \Pr(s_{t+1} | s_t, a_t, \Theta) b_t(s, \Theta), \tag{12}$$

$$b_{t+1}(s, \Theta) = \xi \Pr(o_{t+1} \mid s_{t+1}, a_t, \Theta) \hat{b}_{t+1}(s, \Theta), \tag{13}$$

where ξ is a normalizing constant.

Since we approximate the belief $b_t(s, \Theta)$ by using a set of particles of $\{s, \Theta\}$, the belief can be updated either by a commonly used weighted particle filter [16] or an unweighted particle filter with a Monte-Carlo procedure [36]. In this work we are using the unweighted version.

A common problem for particle filter approaches when dealing with large search spaces is particle deprivation. The existing particles, though distributed, may not cover the real state and get eliminated by contradicting observations. It is possible that all particles are eliminated in this fashion over a short time period. To counter this particle deprivation, usually the set of particles is reinvigorated with new and possibly better matching particles every time an action has been taken. Typically, the new particles are generated by adding artificial noise to existing particles (e.g. Gaussian noise with zero mean on each state value) or by applying a domain specific local transformation to existing particles. We use standard reinvigoration with noise for comparison in our experiments.

Over time, belief update and particle reinvigoration should lead to better matching between particles and true state. However, the reinvigorated particles can be inconsistent with the history of actions and observations. To ensure the consistency of new particles we present history consistent particle invigoration in Section 3.3.1. Additionally, it is possible that the addition of noise and local transformations will still not generate particle in the correct part of the state space. We address this problem in Section 3.3.2 with a novel neural network based method for particle reinvigoration.

3.3.1. History consistent particle reinvigoration

When adding noise or using local transformations on existing particles to reinvigorate the particles in the belief, it is possible that new particles are not compatible with evidence seen before. While the normal belief update generates only particles that are consistent with the prior belief and the used transition and observation models, the use of noise or local operators can reintroduce particles that are in previously excluded areas of the state space. These incorrect particles will (again) be eliminated over time, but they still misguide the planner in the meantime. This problem is strongest when many observations are made that do not carry information (e.g. objects or agents of interest are currently outside the field of view and not observed) as any new particle will agree with such a single observation. To prevent this issue and to distinguish the importance of particles better, we utilize a longer history of past actions and observations to validate and weight the particles.

Algorithm 1 shows how the history of observations and actions

$$h_{j:k} = \{a_j, o_{j+1}, \ldots, a_{k-1}, o_k\} \tag{14}$$

from time step $j > 0$ to current time k is used to ensure that new particles (independently of the chosen reinvigoration method) are compatible with past evidence instead of just the current observation. For each particle to be generated, new parameters Θ are created and then combined with a sampled state s from belief $b_j(s, \Theta)$ at the beginning of the available history. This new particle $\{\tilde{s}_j, \tilde{\Theta}\}$ is used as starting point for simulating the history of actions $a_{j:k-1}$ and recording the resulting states and observations. Then, the likelihood l_t is calculated for the new particle based on the compatibility of the simulated and real observations. The new particle is discarded if the matching is too low. This method reinvigorates the belief with new particles while ensuring consistency with the history. While this method can be expensive in terms of

computation time, it is only executed once per action execution to verify new particle created in the reinvigoration process. In all our experiments the runtime was below 0.5 s. In the future, it has be seen how the computation scales with the history length and there is surely an opportunity to improve this method.

Algorithm 1 History Consistent Particle Reinvigoration.

procedure GENERATEPARTICLE($b_j(s,\Theta)$, $b_k(s,\Theta)$)
  $\{\hat{s}_j, _\} \sim b_j(s,\Theta)$
  $\{_, \hat{\Theta}\} \sim b_k(s,\Theta)$
  $\{\tilde{s}_j, \tilde{\Theta}\} \leftarrow \{\hat{s}_j, \hat{\Theta}\}$ *// e.g. Add noise*
  return $\{\tilde{s}_j, \tilde{\Theta}\}$
end procedure

procedure CHECKHISTORYCONSISTENCY($\{\tilde{s}_j, \tilde{\Theta}\}$, $h_{j:k}$)
  $w \leftarrow 0$
  $t \leftarrow j$
  repeat
    $(\tilde{s}_{t+1}, \tilde{o}_{t+1}, \tilde{r}_{t+1}) \sim \mathcal{G}(\tilde{s}_t, a_t; \tilde{\Theta})$
    $l_t = \Pr(o_{t+1} | \tilde{s}_t, a_t, \tilde{\Theta})$ *// Likelihood*
    if $l_t < \delta$ **then**
      $w \leftarrow 0$
      break
    end if
    $w \leftarrow w + \ln l_t$ *// Log-likelihood*
    $t \leftarrow t + 1$
  until $t > k$
  return $\{w, \tilde{s}_k, \tilde{\Theta}\}$
end procedure

procedure REINVIGORATION($b_j(s,\Theta)$, $b_k(s,\Theta)$, $h_{j:k}$)
  repeat
    $\{\tilde{s}_j, \tilde{\Theta}\} \leftarrow$ GENERATEPARTICLE($b_j(s,\Theta), b_k(s,\Theta)$)
    $\{w, \tilde{s}_k, \tilde{\Theta}\} \leftarrow$ CHECKHISTORYCONSISTENCY($\{\tilde{s}_j, \tilde{\Theta}\}, h_{j:k}$)
    if $w > \epsilon$ **then**
      Add the particle $\{\tilde{s}_k, \tilde{\Theta}\}$ to $b_k(s,\Theta)$
    end if
  until N particles have been added to $b_k(s,\Theta)$
  return $b_k(s,\Theta)$
end procedure

3.3.2. Neural network priors

Initially, the known models and model parametrizations (and the corresponding particles) likely mismatch the actual situation. Therefore, the estimation process depends on the ability of the reinvigoration process to produce new particles that match the actual situation. The performance of the estimation process can be limited by the magnitude of the adaptations in the reinvigoration process. If the magnitude is too small (e.g. local noise), then the new particles could be unable to capture sudden jumps in the dynamics or the mismatch in the agent's strategy. On the other hand, a too large magnitude of the adaptation can disturb the convergence of the particles and decreases the performance of the estimation. To alleviate this problem, in our approach, a recurrent neural network is trained with the past experiences. It is used as input to a mixture density network (MDN) [6] to generate a better guesses on estimation targets. As a result there is no more need to manually determine the appropriate magnitude of the adaptations and sudden jumps in the dynamics can be handled.

We use a recurrent neural network (RNN) to predict different local dynamics and behavior models from the history of observations. The RNN is pre-trained on a dataset collected from various simulations (see Section 4.2 and Section 5.2 for training details).

We use a combination of long short-term memory (LSTM) recurrent neural network [20] and an MDN to predict a multi-modal mixture distribution on the parameter space for Θ. We chose LSTM due to the requirement for storing and utilizing historical information. Instead of using a single LSTM layer, we use a stacked two LSTM layers with expecting that each layer can learn long correlations and short correlations between inputs and outputs respectively. The input to the LSTM is a sequence of observations which can have a various length. For enabling our network to learn a multi-modal probability density over states and time-independent parameters, we use the MDN. The output from LSTM cell at the last time step will be the input to the MDN. The probability density for the MDN output is modeled using a Gaussian mixture model (GMM) of the form:

$$p(s_t, \Theta \mid o_t) = \sum_{k=1}^{K} \alpha_k(o_t)\, \mathcal{N}\big(s_t, \Theta \mid \mu_k(o_t), \sigma_k(o_t)\big), \tag{15}$$

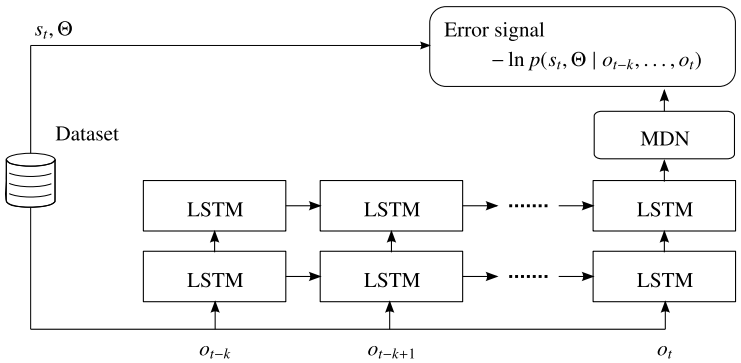


Fig. 3. Training the LSTM-MDN unrolled through time. The LSTM gets histories of observations as input. The LSTM output is used as input to the MDN. The MDN outputs parameters of Gaussian mixture model (GMM) defined over the model parameterizations Θ and state s_t space. The network is trained using the negative log likelihood error signal.

where $\mu_k(o_t)$ and $\sigma_k(o_t)$ are the parameters of the Gaussian components, $\alpha_k(o_t)$ is the mixing coefficient, and K is a hyper-parameter for the number of components. The GMM is used for creating new particles. Note that $\sigma_k(o_t)$ is not a full covariance matrix but only variance. The error signal for the MDN is defined with the negative log-likelihood:

$$E_{\text{MDN}} = -\ln p(s_t, \Theta \mid o_{t-k}, \ldots, o_t). \tag{16}$$

An overview of the training of LSTM-MDN network is depicted in Fig. 3.

Since the output is represented by a GMM, finding the most likely value for the output vector can be considered as a non-linear optimization problem. However, if we assume that the components of the GMM are not too strongly overlapping, we can take the mean μ of the highest component as approximation of the most likely value of (s_t, Θ) as suggested in [6]. Alternatively, we can sample from the GMM. Both methods are evaluated in our experiments.

By using a pre-trained recurrent neural network for generating new particles, the magnitude of the adaptations is no longer constrained and even sudden jumps in the dynamics can be handled. The combination with the history consistent reinvigoration ensures that no contradictory particles are created.

3.4. Combining NN priors and online POMDP solver

Our approach tracks belief over states and model parameters, $b_t(s, \Theta)$, as particles. At each time step, the POMCP algorithm uses MCTS to select the best next action to take based on the current belief. For each particle, it uses a simulator (e.g. a physics engine or an agent behavior simulator) configured with parameters Θ as a black box for the transition and observation model. In the agent behavior simulator (Section 4), the model parameters Θ represents fixed (behavior) types and sensing capabilities (parameters) of other agents. For object manipulation tasks (Section 5), the model parameters Θ configure the physics engine. State s provides the state of the agents or the robot and the objects for initialization. After the chosen action is executed, new observations are used to update the current belief through Bayesian filtering on all particles. Particles that better match the real situation will receive more support and poorly matching particles will disappear.

In addition, the RNN is used to generate new particle suggestions for reinvigoration. The pre-trained RNN (see Section 4.2 and Section 5.2 for training details) receives the history of observations up to the current observation as input. The output GMM provides new potential model parametrizations Θ and states s that are added as new particles to the belief. As it is possible that the RNN will see inputs that are far from any training data, it might produce outputs that do not match the real situation well. In addition, sampling the GMM can result in parameters that contradict the history of observations. Therefore, all suggested particles are checked on consistency with the given history (see Section 3.3 and Algorithm 1) before they are added to the current belief. Thus, the pre-trained RNN is able to adapt the set of considered models effectively given the history and changes the belief state of the POMDP. Finally, in contrast to standard POMCP, the search tree is not pruned as the chance of finding a suitable subtree for reuse is too small. Instead the search tree is discarded and will be rebuilt each iteration.

This process repeats until the task is completed successfully or fails due to particle deprivation. It has to be noted that if the encountered situation is too far away from all training instances then particle deprivation can still occur as the RNN could provide bad suggestions for new particles. If the training data cover the range of possible situations, then the RNN-based reinvigoration helps to prevent particle deprivations because of unfavorable sampling in large state spaces or from dynamics with discontinuities.

An outline of the approach is shown in Algorithm 2 and an overview of the components of the system is depicted in Fig. 1.

Algorithm 2 Proposed method.

procedure SEARCHANDEXECUTE($b_k(s, \Theta)$, h_k)
  $t \leftarrow k$
  repeat
    Construct/update the search tree $T(h_t)$ from simulations starting from $\{s_t, \Theta\} \sim b_t(s, \Theta)$
    Execute planned action a_t in the world
    Receive observation o_{t+1} from the world
    Update the belief $b_{t+1}(s, \Theta)$
    Perform history consistent particle reinvigoration:
    repeat
      $\{s_{t+1}, \Theta\} \sim$ RNN$(o_k, \ldots, o_{t+1})$ (see Section 3.3.2)
      $\{w, s_{t+1}, \Theta\} \leftarrow$ CHECKHISTORYCONSISTENCY$(\{s_{t+1}, \Theta\}, h_{k:t+1})$
      if $w > \epsilon$ **then**
        Add the particle $\{s_{t+1}, \Theta\}$ to $b_{t+1}(s, \Theta)$
      end if
    until N particles have been added to $b_{t+1}(s, \Theta)$
    $t \leftarrow t + 1$
  until *Task is completed* **or** *Out of particles*
end procedure

4. Level-based foraging domain

We evaluate our approach in the level-based foraging domain [1,2] that is a test domain for *ad hoc teamwork* [39]. Here, the aim of the experimental evaluation is to answer the following questions:

1. What is the performance of our approach on estimation of other agent's behavior type and their parameters, and on task completion ratio compared to the baselines?
2. How does our approach work when availability of the informative observations becomes sparse after introducing new partial observability?

4.1. Experimental setup

We evaluate our approach in the level-based foraging domain [1,2]. In this domain, a team of agents needs to collaborate to collect items from a grid. Each agent and item has an individual level which is in the range $[0, 1]$. Each agent is capable of moving in four directions (actions N, E, S, W) and Load to attempt to pick up the object. A team of agents can collect an item if (i) the agents are next to the item, (ii) they choose the load action, and (iii) the sum of their levels is greater than the item's level. The team receives a reward of $+1$ when an item is collected. The reward is 0 in all other cases. A single agent is under our control. Action probabilities for other agents are assigned based on their current destination which is determined from both the agent's type and their observations determined by their view cones. The agent types $d_j \in \{d^{L1}, d^{L2}, d^{F1}, d^{F2}\}$ are a distance-based leader, a level-based leader, a distance-based follower and a level-based follower respectively. The four types have different routine:

- d^{L1}: if sees items, chooses furthest item, else moves randomly.
- d^{L2}: if sees items, chooses item with highest level (below own level if exist), else moves randomly.
- d^{F1}: if sees agents but no items, chooses furthest agent; if sees agents and items, chooses item that furthest agent would choose as type d^{L1}; else moves randomly.
- d^{F2}: if sees agents but no items, chooses agent with highest level above own level, or furthest agent if none are above own level; if sees agents and items, chooses item that agent, who was chosen with the same manner before, would choose as type d^{L2}; else moves randomly.

These agent types are as defined in [2].

The destinations are stored as internal states until they are reached. A^* [18] is used to find the shortest path to reach to the destination. The view cones are parametrized with the view radius $p_2 \in [0.1, 1]$ that is multiplied with the grid world size, and the view angle $p_3 \in [0.1, 1]$ as $2\pi\, p_3$. Here, while other agents have view cones, the agent under our control (colored blue in Fig. 4) can see entire grid (location of other agents and both location and level of all objects.). Note that in Section 4.5, we introduce the new partial observability by considering our agent's view cone shown as blue-colored in Fig. 4. The capabilities (parameters) of the other agents are unknown. The aim of this domain is for our agent to select collaborative actions to collect items with other agents while reasoning about the agent's types d_j, skill level $p_1 \in [0, 1]$ and view cone parameters $\{p_2, p_3\}$ based on their observed actions.

We use POMCP with root parallelization [7] with 8 threads on a standard laptop with a quad-core CPU with hyper-threading. The number of rollouts in each time step is 500. As in [36], we introduce domain knowledge to the search algorithm by defining a set of *preferred actions* that the rollout policy selects uniformly from. When using preferred actions, if another agent is staying next to an item for more than two time steps, then our agent prefers to move in directions of that item. We run experiments in the domain with 3 agents and 10 items in a 15×15 world. We performed 100 simulations

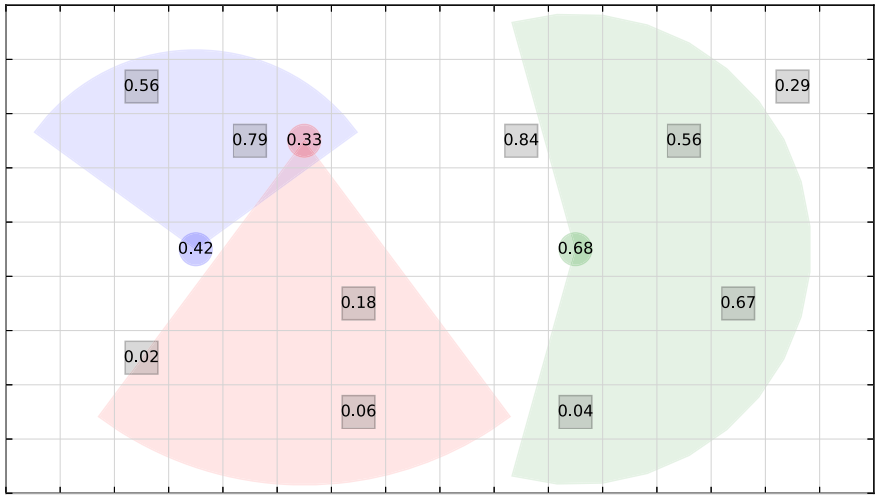


Fig. 4. Level-based foraging domain. Colored circles are agents (blue is the agent under control) and grey squares are items. Skill levels are shown inside agents and items. The colored regions show the other agent's view cones. Our agent can see either entire grid or the area within its view cone. The view cone for our agent is added to introduce new partial observability to the domain (see Section 4.5). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

from randomly initialized states, agent types and parameters. The maximum number of time steps is 150. The discount factor is $\gamma = 0.95$.

4.2. Data collection and NN training

We collected data from experiences in multiple simulated environments to train the RNN that estimates other agent's behavior type and the parameters of sensing capabilities. Here, we assume that 3 agents and 10 items exist in a 15×15 grid world. Although there are two other agents in the grid world, we train a single RNN using the target agent coordinate frame so that it is applicable across them. The input for the RNN is a variable length sequence of observations each consisting of (i) 6 continuous values for the positions and orientations of two other agents, (ii) 30 continuous values for the positions and skill levels of 10 items, and (iii) 10 binary values to represent items collected. All positions and orientations are transformed to the target agent coordinate frame. Note that the two other input agents here are our own agent and the second teammate (that is not the current target agent). Here, we use the elapsed time steps from the initial time step as the length of the sequence. The output is a GMM over the type and the parameters of the target agent $\Theta_j = \{d_{type}, p_1, p_2, p_3\}$. We use *one hot representation* for a discrete variable output from the NN. We train a single RNN, and apply it to predict the behavior of each teammate.

We collected the data of 100 k instances from simulations. By using these instances, we generate the various length time series data for training (500 k). We train our network of two LSTM layers which receive variable lengths of inputs, with 128 nodes each and a tanh activation together with a MDN output that contains 4 Gaussian components (LSTM-MDN). As references, we also train two other networks; One is LSTM-MSE trained with the mean squared error, and the other LSTM-Classifier is trained with the soft max cross entropy. All of the networks have identical hidden layers (two LSTM layers with 128 nodes each). We used a batch size of 128 and Adam optimizer with fixed learning rate of 0.001 in our training.

4.3. Baseline

To put the performance of our approach in perspective, we use a state-of-the-art method as baseline for comparison. Albrecht and Stone [2] investigated 9 methods that are combinations of 3 approaches of selecting types for parameter update and 3 approaches for estimating parameter values. The 3 approaches of selecting types for parameter update are: updating all types (*All*), sampling a single type from the belief, and sampling a single type using a bandit algorithm. For the estimation of parameter values, they use: Approximate Gradient Ascent (AGA), Approximate Bayesian Updating (ABU), and Exact Global Optimization (EGO). AGA updates parameter estimates by fitting a polynomial to a set of samples, and following its gradient. ABU uses polynomials to approximate the probability with which action is chosen and compute the product of that and the current belief, and finally perform Bayesian updates. By using Bayesian optimization that efficiently selects a new parameters to evaluate, EGO estimates parameters directly with the target function that can have discontinuities instead of relying on continuous polynomials.

We take the results reported for the best combinations of methods from [2] as a baseline for the different tests. We will evaluate:

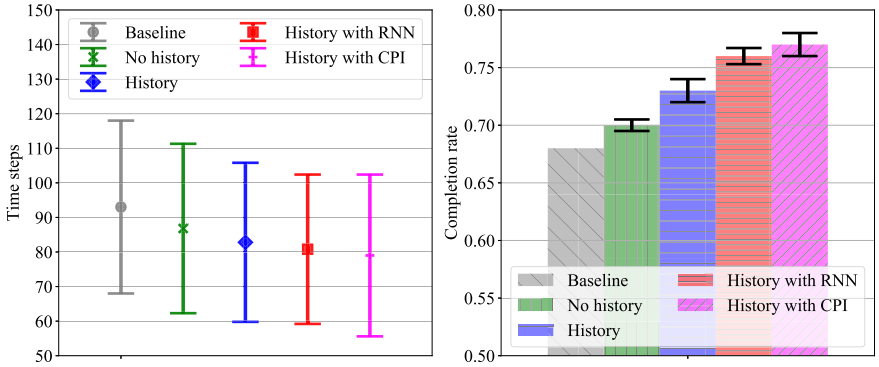


Fig. 5. Required time steps (means and standard deviations) to complete the task and completion rates in the level based foraging domain with 3 agents and 10 items in a 15×15 world. Results are averaged over 100 instances. The performance of *All*+EGO from [2] is used as a baseline and shown as leftmost grey colored line/bar.

1. Time steps required in completion and completion rates: Here we use *All*+EGO as baseline as it had the highest completion rate and lowest average time steps.
2. Type and parameters estimation: For parameters we use *All*+ABU as baseline. It performed best for estimation of skill level and was close with *All*+EGO for view radius and angles. For type estimation, we chose *All*+EGO as it performed much better. As they reported two values for different number of points used inside the EGO method, we use the better performance for 50 points with a mean error of 0.42. We use the average estimation result at the end of the trials and show it as a constant dashed line in the corresponding plots.

In addition to the baseline, we also test the performance of our approach if the belief receives an injection of a correct particle at each time step. As perfectly matching particles are guaranteed to always exists, this combination provides a good comparison to show the best possible performance when there is (almost) no estimation error. This combination is denoted as (*History with CPI*) in the following sections.

4.4. Results

Fig. 5 shows the average number of time steps needed to complete the task and the completion rates in a 15×15 world with 3 agents and 10 items. The completion rate is the percentage of instances which were completed successfully within the given maximum time steps. The average time steps are only for successful instances.

It can be seen that there are only small differences in the number of required time steps though our approach can slightly reduce the average number. It has to be noted that the standard deviation is very large for all cases and the improvement is not statistically significant in this metric. A similar issue could be seen in [2]. We attribute this to several issues: 1) Different problem instances can require substantially different number of steps for optimal solution. This results in generally high variance for the required time steps of each method. 2) Only successful trials were included in this evaluation and successful solutions are all very similar for each specific problem. Here Albrecht and Stone [2] also noted that the MCTS planner is robust enough to "absorb" differences by often taking similar actions despite different belief. This results in small differences in the means of the required time steps.

With regards to the completion rate our approach achieved substantial improvements compared to the baseline. The achieved completion rate even gets close to the performance with Correct Particle Injection (*History with CPI*). *All*+EGO from [2] was used as baseline and could achieve completion rate of 68%. Standard POMCP with noise-based reinvigoration and without history (*No history*) achieved 70%. By using history consistent reinvigoration (*History*) the completion rate rose to 73%. It would be expected that this method would perform similarly to the baseline as they use similar planners and both respect the history of observations. Part of the difference in performance could be attributed to the use of *preferred actions* that could provide a slight edge here. Nonetheless, the RNN based sample injection (*History with RNN*) can increase the completion rate to 76%. As a comparison, we also show the performance of the system with perfect reinvigoration (History with CPI; Correct Particles Injection) for which the belief receives an injection of a correct particle at each time step, and it achieved 77% completion rate. The performance can be seen as the best possible performance of the system having particle reinvigoration. Note that the task cannot be completed when two teammates simultaneously commit to the different high level items if each item can be collected only if all three agents collaborate. Also, other agents that have a small view cone often extend the needed time past the limit because they tend to wander long time steps until finding items. Those kinds of situations can be handled by introducing a communication among the agents, but that is not focus of this paper.

Fig. 6 shows the mean error and standard deviation in the type and parameter estimation. It can be seen that the standard deviation is very large for all methods and error types. This is to be expected as the amount of information available varies for each problem instance: e.g. if another agent fails to pick up an object of difficulty 100, then this contains

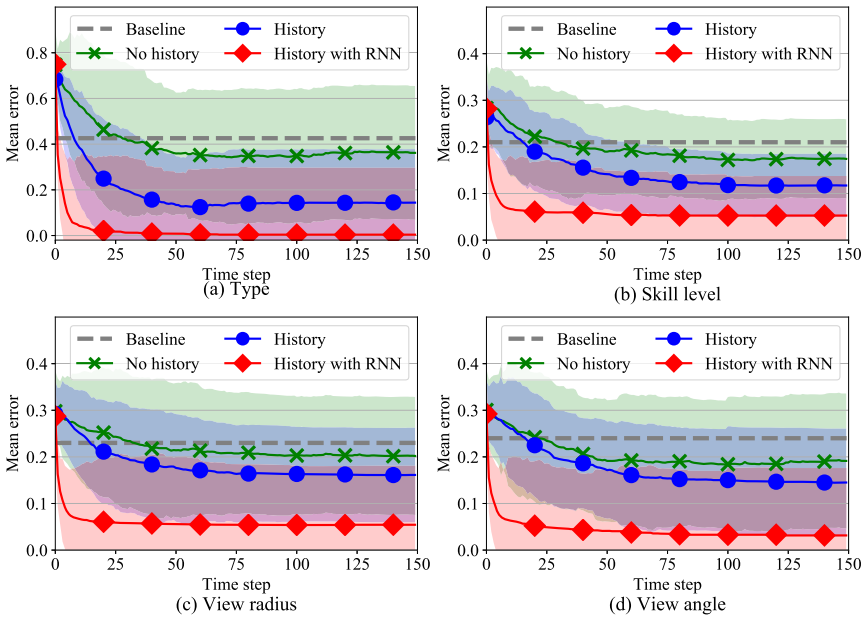


Fig. 6. Type and parameters estimation error (lines) and standard deviation (shaded areas) in the level based foraging domain with 3 agents and 10 items in a 15×15 world. The estimation error for type at time t is defined as $\left|1 - \Pr(d_k^*|h_t)\right|$ for the agent k's true type d_k^*. The estimation error for parameters at time t is defined as the mean absolute error $\sum_i^N \left|\hat{p}_i - p_k^*\right|/N$, where N is the number of particles and p_k^* is true parameter for agent k. Results are averaged over 100 instances and both other agents. The mean error from [2] at the end of the trial is used as a baseline and shown as a constant dashed line: The best performance of *All*+EGO is used for type estimation (a) and *All*+ABU is used for parameter estimation (b,c,d).

almost no information. As a result, there will often be a high estimation error that is not due to the approach but based on the available information. Nonetheless, our approach substantially improved the mean estimation error for all parameters compared with the baseline. The improvement is especially strong in the type and skill level estimation. For type estimation the baseline achieved a mean error of 0.42 while (*History with RNN*) lowered this to 0.02. The mean error for the estimation of skill levels went from 0.22 down to 0.06. The improvements for view radius and view angle estimation were similar. This improvement is a result of the combination of the history consistent particle reinvigoration and the pre-trained RNN. The history consistent particle reinvigoration in our belief update that can ensure the compatibility of new particles with evidence seen before. And the pre-trained RNN appropriately suggests new particles in good areas of the state space.

The RNN prediction errors is shown in Fig. 7. For the LSTM-MDN prediction, we use the mean value of a Gaussian component which has highest mixing coefficient $\hat{\alpha}_{k \in K}$ among K components for each time step. Here, $K = 4$. Around the initial time steps, the type prediction by LSTM-MDN achieves better performance compared to others. This is due the multi-modal outputs for the networks. Specifically, LSTM-MDN generates the almost equal values for the mixture coefficients around these time steps. Since MSE and classifier are designed for the unimodal data, their performances are worse. As time progresses, all three achieve the correct predictions because they get longer length of the inputs and that reduce the possibility of the multi-modal outputs. This effect of the input length can be seen for other parameter predictions as well. Note that the performance of the RNN in each instance is dependent on the information in the provided observations. As it is possible for observations to contain little information in this domain (e.g. an agent waits a long time next to an item with high skill level), the mean error of the RNN prediction can, as expected, also be very high until more information is received.

4.5. Adding new partial observability

In order to increase the difficulty of the level-based foraging domain while also adding a constraint that a real agent would have to handle, we extend the level-based foraging domain with partial observability for the controlled agent. This is done by adding a view cone to our agent while the original work assumed that the agent can see the entire grid [2]. The view cone is parametrized with the view radius $p_2 \in [0.1, 1]$ and the view angle $p_3 \in [0.1, 1]$ as for the other agents. We assume that our agent can observe other agent's locations and their actions only when they are in this view cone. As with [2], we assume our agent already knows the locations and levels of all items in a grid. However, each item has to be seen in order to know if it has been collected. As an example, for the situation depicted in Fig. 4, the blue agent knows the location and level of all items, it knows if object 0.31 and 0.29 have been picked up, and it knows the location of the red agent. It has no information on the pickup status of all other objects or the location of the green agent. Moreover, our agent sometime observes only the fact that the item has already been collected, but not who picked it up. For such cases, it is expected to reason about other agent's skill level and its past actions without knowing the current location of the agent.

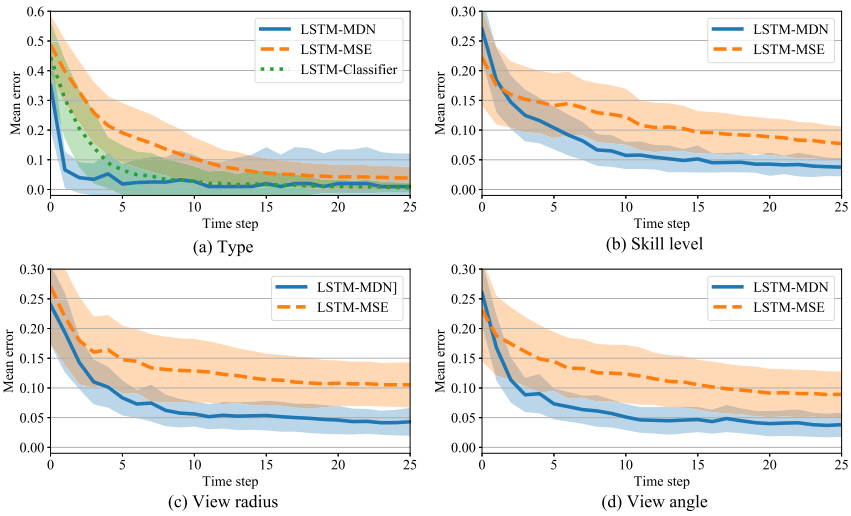


Fig. 7. RNN prediction error in type and parameter estimates (lines) and standard deviation (shaded areas) for another agent in the level based foraging domain. RNN predictions at time step t are $\{\hat{d}_{type}, \hat{p}_1, \hat{p}_2, \hat{p}_3\} \sim$ RNN $(o_1, \ldots, o_t)$ for LSTM-MDN and LSTM-MSE, and $\hat{d}_{type} \sim$ RNN $(o_1, \ldots, o_t)$ for LSTM-Classifier. The estimation error for type at time step t is defined as $\left|1 - \Pr(\hat{d}^*_{type})\right|$ for the true type d^*_{type}. The estimation error for parameters at time step t is defined as $\left|\hat{p}_i - p^*_i\right|$. $p^*_{i\in\{1,2,3\}}$ are true parameters. Results are averaged over 100 instance.

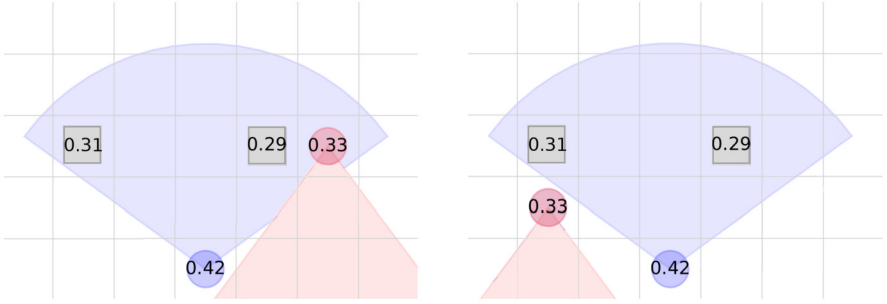


Fig. 8. Examples in the level-based foraging domain with added partial observability: our agent (blue) can only perceive the location and actions of other agents within its view cone. (Left) If the red agent picks up the box, this action can be observed. (Right) If the red agent picks up the box, the blue agent only sees that the box disappeared and can only hypothesize who picked it up.

Such a scenario can be seen in Fig. 8. On the left the blue agent can see both the object with level 0.29 and the red agent next to it. If the object is picked up, this information can directly be associated with the capabilities of the red agent as it can be seen performing the action. On the right side, the blue agent can see the object with level 0.31 but not the red agent. If the object is being picked up, the blue agent can only hypothesize that the red agent might have been the one to pick it up.

4.6. Results – extended partial observability

As expected, in comparison to the original domain, the performance for completion rate and estimation quality decreases as the introduced partial observability strongly reduces the amount of information that the agent is receiving and overall increases the difficulty of the task. Fig. 9 shows the average number of time steps needed to complete the task and the completion rates. Similar to the results in the normal domain (Sec. 4.4), the mean number of required time steps could be lowered slightly, but the variance is high and the change is not statistically significant. While the standard POMCP (*No history*) achieved 50% completion rate, the history consistent particle reinvigoration (*History*) achieved 58%, and the RNN based sample injection (*History with RNN*) achieved 60%. As a comparison, we also show the performance of (*History with CPI*) (with correct particles injection), and it achieved 70% completion rate. They complete the task in fewer time steps while having similar magnitude of standard deviations with the baseline. Fig. 10 shows the mean error in the type and parameter estimation over the number of steps. Our approach improves the estimation quality for all parameters compared with the standard POMCP (No history). While the improvement from the effects of the history consistent particle reinvigoration is comparable to the original level based foraging domain (Fig. 6), the benefit from using the RNN for reinvigoration is

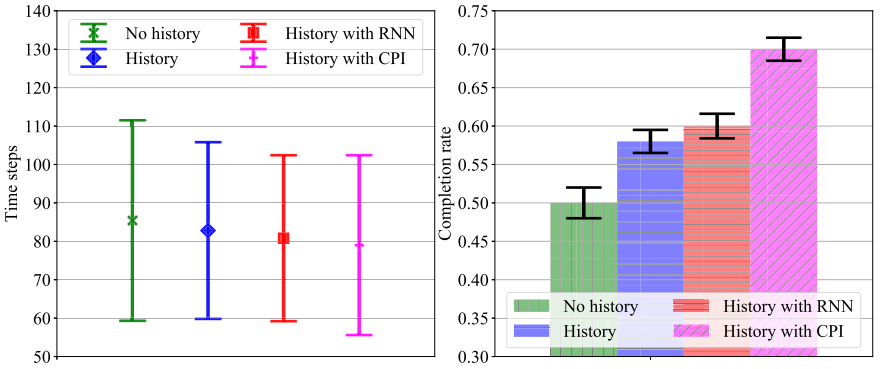


Fig. 9. Required time steps (means and standard deviations) to complete the task and completion rates in the *partial observable* level based foraging domain with 3 agents and 10 items in a 15×15 world. Results are averaged over 100 instances.

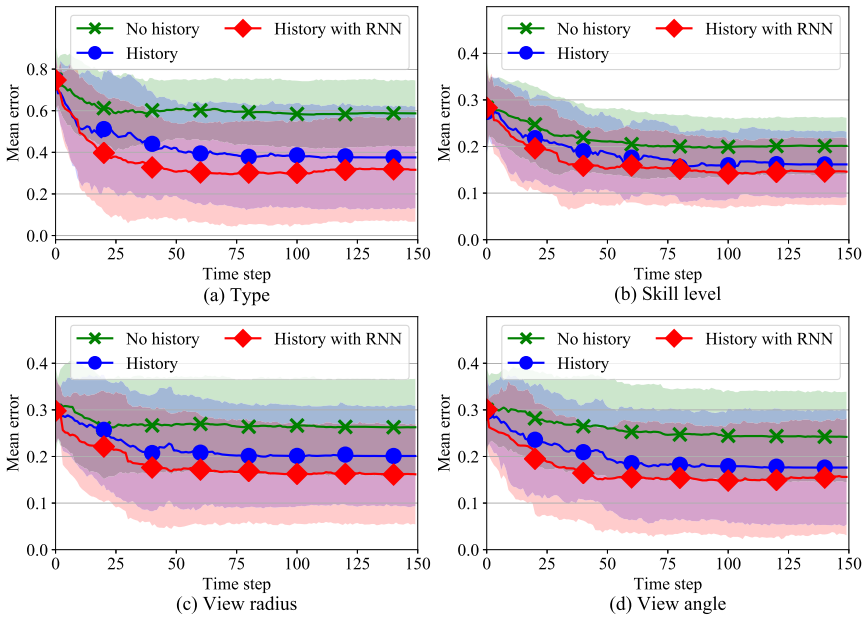


Fig. 10. Mean error (lines) and standard deviation (shaded areas) in type and parameter estimates for the *partial observable* level based foraging domain. Results are averaged over 100 instances in each 15×15 world (with 3 agents and 10 items).

diminished. This is because we reused the pre-trained RNN for the fully observable version, but it can only predict when our agent can see the other agent(s) and in general receives input sequences with less information.

5. Object manipulation domain

The proposed approach is not limited to behavior and parameter estimation in the multi-agent setting but can be applied to many problems with partial observability, uncertainty, and model adaptation. In this section, we evaluate our method on a domain of robotic pushing tasks both in simulation and on a real robot. The aim of the experimental evaluation is to answer the following questions:

1. Does our approach enable adaptation of the mismatched world models to the encountered situation?
2. Can our approach handle dynamics with discontinuities?
3. How does our approach scale with the complexity of the world models?

5.1. Experimental setup

We evaluate our method on various robotic pushing tasks with increasing complexity. These tasks involve controlling a robotic arm with an end effector to push an object to the goal (Fig. 11). We use several task settings with increasing

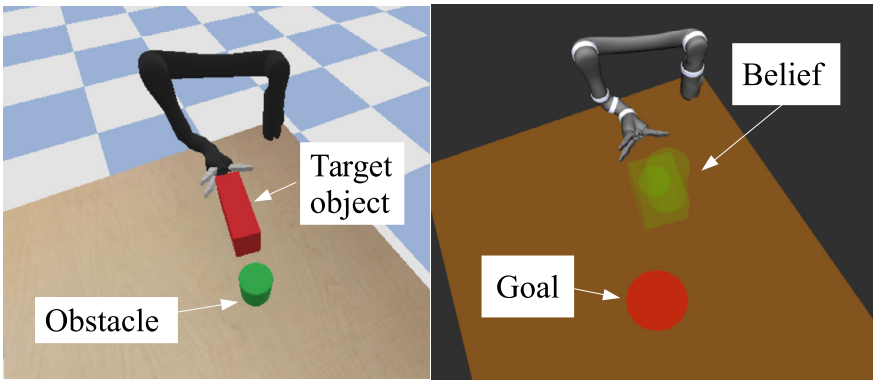


Fig. 11. Task for the evaluation: Pushing a target object to the goal location. The shape of the target object (box, cylinder, or sphere) is unknown to the robot. The existence of obstacles around the target object is also unknown.

Table 1
Number of dimensions of variables used for experiments with Sawyer arm.

	Symbol	Type	D_{total}	N_{total}	$\boldsymbol{q}$	F/T	$\boldsymbol{x}_{pose}$	$\boldsymbol{v}$	$\boldsymbol{x}_{pose}^{obs}$	$\boldsymbol{v}^{obs}$	Θ
Input to planner	o_t^d	Discrete	–	256^6	–	3	3	–	–	–	–
Input to RNN	o_t	Continuous	20	–	7	6	7	–	–	–	–
Belief	$b_t(s, \Theta)$	Hybrid	47	–	7	–	7	6	7	6	14

Table 2
Number of dimensions of variables used for experiments with Jaco arm.

	Symbol	Type	D_{total}	N_{total}	$\boldsymbol{q}$	$\ddot{\boldsymbol{q}}$	$\boldsymbol{x}_{pose}$	$\boldsymbol{v}$	$\boldsymbol{x}_{pose}^{obs}$	$\boldsymbol{v}^{obs}$	Θ
Input to planner	o_t^d	Discrete	–	256^6	–	3	3	–	–	–	–
Input to RNN	o_t	Continuous	13	–	–	6	7	–	–	–	–
Belief	$b_t(s, \Theta)$	Hybrid	46	–	6	–	7	6	7	6	14

complexity that use combinations of three types of geometric shapes (box, cylinder, or sphere) with unknown sizes for the target object and existence of a movable or static obstacle around the target object. The existence of the obstacle provides dynamics with discontinuities. The robot has to plan actions based on real observations while validating the models across the assumed environments.

We use two different robot platforms: the first set of experiments uses a simulated 7 degrees-of-freedom (DOF) Rethink Sawyer arm with a simplified ball-shaped effector. The second set uses both a simulated and a real Kinova Jaco arm with 6 DOF and a 3-fingered hand. Sensor feedback is obtained from a force-torque sensor on the wrist and measured joint torques, respectively. For simulations, we use the Bullet physics engine [9].

To evaluate our approach, we compare our method against standard POMCP. This algorithm is state-of-the-art for POMDP planning and can handle very large state and observation spaces. For the reinvigoration process that happens after each taken action and new observation, we use artificial noise on individual parameters as described in Section 5.2.

Actions: We define 10 discrete actions for relative movement of end effector positions: 8 actions for the xy-plane (forward/backward, right/left, and four diagonals), and up/down along the z-axis within 2 cm. Each movement is executed with a minimum jerk trajectory with 20 time steps.

Rewards: the robot arm receives a reward of -1 at each step, -5 for a large force or joint torque (e.g. 100 N for the force measured at the wrist joint or 30 N m on each joint), -5 if the arm exceeds the reachable region, $+10$ for moving the object, and $+100$ for the object reaching the goal region. The discount factor is $\gamma = 0.95$.

Observations: used differently as input to the planner and to RNN: Prediction by the RNN is computationally cheap and can handle continuous values without problem and achieve better quality. As planning by MCTS is computationally expensive, discretization keeps it computationally tractable:

1) Input for the RNN: 20 continuous observations with 7 joint positions, 6 measured forces and torques on the wrist, and 7 for the pose of the object for Sawyer. 13 continuous observations with 6 measured joint torques and 7 for the pose of the object.

2) Input for the POMDP planner: only a subset of all observations is used and coarsely discretized. This subset consists of 3 measured forces and the location of the object with the Sawyer and 3 measured joint torques (distal joints) and object pose. Inputs for the RNN and POMDP planer are shown in Table 1 and 2.

Table 3
Assignments of probabilistic distributions for parameter sampling.

States and properties	Probabilistic distribution
$\boldsymbol{x}_{pose}$, $\boldsymbol{x}^{obs}_{pose}$	Truncated normal (μ, σ, a, b)
m, m^{obs}	Gamma (α, β)
$\boldsymbol{x}_{com}$, μ_c	Truncated normal (μ, σ, a, b)
$\boldsymbol{p}_{size}$, $\boldsymbol{p}^{obs}_{size}$	Truncated normal (μ, σ, a, b)
d_{type}, d^{obs}_{type}	Categorical (p)

The discretized observation i_o is calculated by standardizing and binning o, using a range of (a, b) and $N = 255$ (8-bit) bins with

$$i_o = \left\lfloor \frac{(b-a)\, o}{N} + a + 0.5 \right\rfloor, \tag{17}$$

where $N = 255$ (8-bit) for $(a_{loc}, b_{loc}) = (-10, 10)$ [m], $(a_{ori}, b_{ori}) = (-2\pi, 2\pi]$ [rad], $(a_f, b_f) = (-500, 500)$ [N], and $(a_t, b_t) = (-160, 160)$ [N m]. Here, theoretically, we have $(N+1)^6$ possible observations however, only a subset of them are encountered inside of the online planner because it evaluates only the current belief and associated states and observations.

5.2. Data collection and NN training

We collected data from physical interaction experiences in multiple simulated environments to train the neural network prior. Here the input to the RNN are continuous observations o_t and the output is a GMM over (s_t, Θ) as described in Equation (15). The state s consists of 6 or 7 joint positions of the arm plus 26 continuous values for the poses and the velocities of the target object and the obstacle. The physical parameter Θ consists of 12 continuous and 2 discrete values.

The dataset consists of data from the simulations of the pushing task in randomized worlds. For this, we sample the full physical parameters and initial states by using univariate probabilistic distributions. Table 3 shows our assignments of probabilistic distributions to each parameter for sampling. Additional information can be found in Appendix A. The object pose is initialized with $\boldsymbol{x}_{pose}$. Mass m is a positive value and the friction coefficient μ_c is in the range $[0, 1]$. The parameters $\boldsymbol{p}_{size}$ represents the sizes of the object such as the radius and height for the cylinder primitive. The location of the center of mass $\boldsymbol{x}_{com}$ is sampled within the ranges of the sizes for each dimension. The geometric types are specified as $d_{type} \in$ {box, cylinder, sphere}, and obstacle types $d^{obs}_{type} \in$ {box, cylinder, sphere, *none*}. Note that the object type for the obstacle can be *none* to represent the situation without an obstacle. In order to generate worlds in the physics engine, we sample the parameters by using the assigned distributions.

Since mass is in the range $[0, +\infty)$ and it is reasonable to assume most objects are not enormous heavy, we use a gamma distribution as a right skewed distribution. A lognormal distribution can be used alternatively if a heavier tailed distribution is more preferred. A categorical distribution is reasonable to model an index (e.g. object type) that can take one from finite numbers of categories. For other continuous values, we use truncated normal distributions as efficiently model uncertainties while considering intervals.

Once a randomized world is generated in the physics simulator, the robot must interact with an object to gather experiences. To select meaningful and efficient actions instead of random actions, however, the robot needs a policy for that purpose. In this paper, since we use a physics engine and can access the resultant state of each action directly, a simple heuristic policy is used. The policy is based on the Euclidean distance between the end effector x_{ee} and the object center x_{oc}. Here, since our state s consists of the arm joint positions and the poses and the velocities of the target object and the obstacle, the x_{oc} is a part of the state and we can compute the x_{ee} by using forward kinematics. The Euclidean distances are computed for the resultant state s' of each action $a \in \mathcal{A}$ and the policy selects the action(s) to minimize the distance. Thus, a set of optimal actions for the given state s is computed as:

$$\pi_{\text{interact}}(s) = \arg\min_a \, \| x_{ee}(s, a) - x_{oc}(s, a) \|, \tag{18}$$

subject to:

$$a \in \mathcal{A}, \{x_{ee}, x_{oc}\} \leftarrow s' \sim f(s, a). \tag{19}$$

Examples of the sampled worlds are shown in Fig. 12.

We collected the data of 15k transitions from simulations. By using this dataset, we trained our network of two LSTM layers with 128 nodes each and tanh activation together with a MDN output with 10 components. We used a batch size of 128 and Adam optimizer with fixed learning rate of 0.001 in our training.

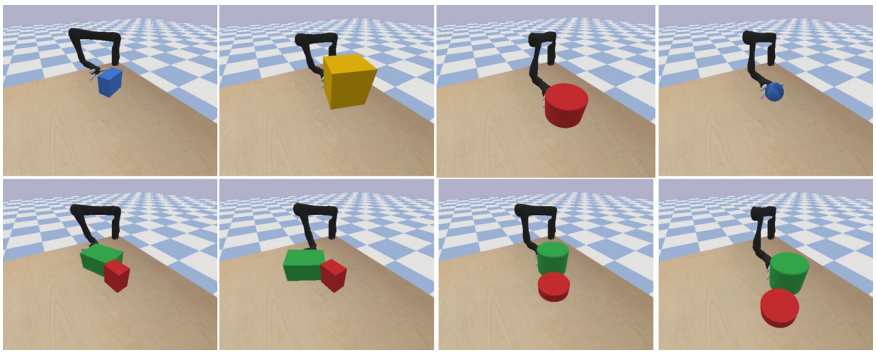


Fig. 12. Data collection in simulation: snapshots of the physical interaction experience. The robot follows a heuristic policy that minimizes the distance between the hand-tip and the center of the target object.

Table 4
Comparison of success rate of methods on the simulated tasks.

Methods	D1	D2	D3	D4	D5
Standard POMCP	**24/30**	16/30	10/30	1/30	0/30
Proposed w/max. value	12/30	6/30	6/30	2/30	0/30
Proposed w/sampling	20/30	**17/30**	**15/30**	**10/30**	**7/30**

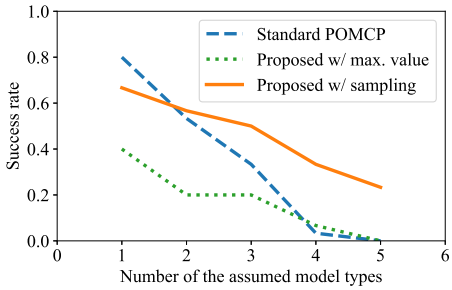


Fig. 13. Success rate of the approaches for the different settings D1-D5.

5.3. Comparisons in simulation

To evaluate our approach, we compare two variants against POMCP with standard reinvigoration. These variants differ slightly in the way they merge the RNN outputs into current belief: 1) The most likely value (mean μ of the highest component) of the GMM is selected, 2) the GMM is sampled. The suggestions of the RNN are added as new particles to the belief.

Tests are performed in five settings (D1-D5) with increasing complexity. In the simplest case (D1), the target object is known to be of type *box* and there cannot be any obstacles. In settings D2 and D3, the object can be any of two (*box, cylinder*) or three types (*box, cylinder, sphere*) respectively. D4 adds a possible static obstacle with unknown shape and properties. For D5 the obstacle can also be movable.

Each setting was tested 30 times with each approach. The results can be seen in Table 4 and Fig. 13. For the simplest case D1, standard POMCP works best because it focuses on adjusting a single prediction model, meaning the method does not generate particles for other models. In contrast, in our approaches the RNN can generate the particles broadly based on prior experience, meaning for all prediction models, making its problem more difficult. The RNN is pre-trained on the dataset of various situations from setting D5 (see Section 5.2 for training details) and as such also considers cases from D2-D5 when predicting for D1. Thus it can also propose models for object types and parametrizations that are not available in D1, but that are still consistent with the observations. Our result indicates that such non-helpful particle suggestions by the RNN hinders the search of the POMDP solver for this simplest case. However, it can handle increased complexity much better than standard POMCP. Standard POMCP does not work well for cases D2-D5 because its particle reinvigoration is inefficient for higher number of prediction models. Additionally, it is difficult to find appropriate new particles if the dynamics has discontinuities. Our methods have higher chances of drawing new particles far from current particles because they are trained with past experience and based on current observations. By guiding the belief update globally based on encoded experiences using the RNN, in addition to local search based on artificial noise which is used in the standard Bayesian filtering, our approach can find more reasonable particles even they are far from the current distribution of particles. Thus

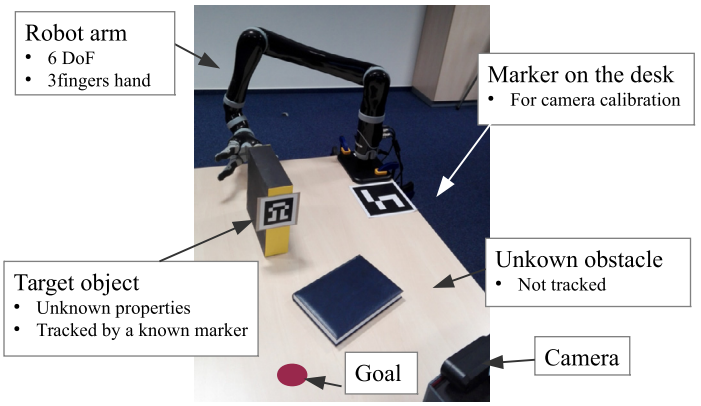


Fig. 14. Experimental setup: A Kinova Jaco arm pushes an object with unknown properties to reach a goal area on a table. A movable or a static obstacle can be in the scene, although it is not tracked by the sensor.

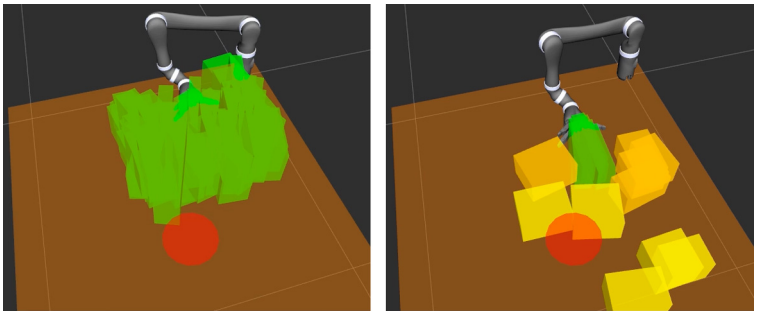


Fig. 15. Example of an evolution of belief over states and physical parameters in the presence of an obstacle: the particles for the target object (green), the particles for the obstacle (yellow) and the goal area (red). (Left) An initial belief does not consider obstacle's existence. (Right) A belief updated to explain real observations of the object pose and the arm joint torques.

the approach improves the performance of model adaptation and the scalability against number of the prediction models. Also, the comparison of the two variants of our approach shows that stochastic use of prior knowledge works better than deterministic use.

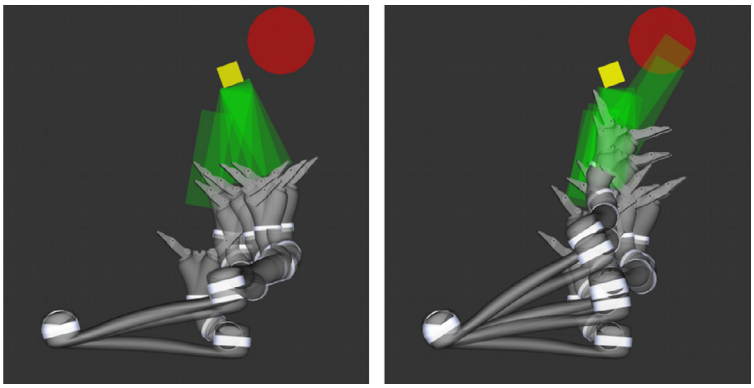
5.4. Robot experiment

We use the Kinova Jaco to evaluate how the approach performs on a real robot. Fig. 14 shows the experimental setup. In the real robot experiment we are focusing on situations that exhibit dynamics with discontinuities. Therefore, we use a mix of settings D1 and D5 giving more importance to the existence of obstacles and ignoring uncertain object types. Again the robot needs to push the target object to the goal region. The object is tracked with a marker. A movable or static obstacle can be in the scene, although it is not tracked. An initial belief over the states and physical parameters is given but with a mismatch to the real scene: the object type (box) is known, but all other properties and the existence of an obstacle are unknown. Again, the existence of the obstacle provides dynamics with discontinuities.

The RNN is pre-trained with 20k transitions collected from various simulations as described in Section 5.2 and Fig. 12, but with a Jaco arm model. (Note that the fingers always remain open in our experiments.) Fig. 15 shows how the belief is update with the RNN based method to explain the observations received when impacting the obstacle. Fig. 16 shows a sequence of executed actions from the same initial belief by two methods. While standard POMCP stops after contacting with the obstacle due to the lack of the particles to explain the situation by only adding Gaussian noise (Fig. 17), our method successfully generated suitable particles to represent the obstacle and was able to choose actions to get around the obstacle (Fig. 18). We conducted 30 experiments in simulation and with the real robot. The results can be seen in Table 5. We found slight decreases of success ratio on both methods on a real robot compared to simulations; however, our methods still handles discontinuities in the dynamics much better than POMCP.

6. Conclusions

In this paper, we presented an approach for adaptive decision making that combines an online POMDP planner with a pre-trained RNN and simulators as the source of priors for different prediction models. We use a pre-trained LSTM in combination with a MDN to generate predictions of model parametrizations and states that are used to extend our



(a) Standard POMCP (b) Proposed w/ sampling

Fig. 16. Overlaid sequences of executed actions from the same initial belief. The task is to move a box (green) to the goal region (red). The initial belief is not assuming the static obstacle behind (yellow). While the standard POMCP stopped after contacting with the obstacle due to the lack of the particles to explain the situation by only adding Gaussian noise, our method succeeded by generating suitable particles from pre-trained RNN.

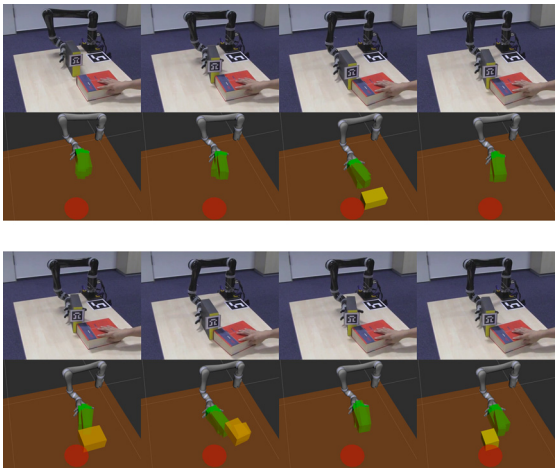


Fig. 17. Standard POMCP: The robot arm performs a pushing task. The initial belief does not consider the static obstacle (red book support by a human hand) in the real scene that blocks the direct path to the goal region. After making contact with the obstacle at step 2, no suitable particles are generated that explain the observed resistance while moving the box. Though some particles have obstacles, their parametrization is poorly matching and the particle gets quickly eliminated again. As the robot fails to model the existence of the obstacle, the robot keeps trying to push through the obstacle and fails to reach the goal.

Table 5
Comparison of success rate for simulated and real robot.

Methods	Simulation	Real robot
Standard POMCP	10/30	7/30
Proposed w/ sampling	**25/30**	**23/30**

hypothesis space. As a result the particle reinvigoration process can leverage prior experience to produce better particles to update and adapt the belief. Proposed particles are validated to ensure that they are consistent with the history of the observations and actions. These enhancements to the particle reinvigoration process help alleviate problems arising from poor sampling quality in large state spaces and enable handling of dynamics with discontinuities. In addition, the model parametrizations are used to configure a physics engine that functions as a black box for transition and observation models that can be used by the POMDP planner.

We evaluated our approach in detailed experiments in a cooperative multi-agent domain and in a single agent object manipulation domain. In the level-based foraging domain the proposed approach achieved substantial improvements in task completion rates and estimations of other agents behavior type as well as their parameters compared to a state-of-the-art method. In addition to the standard domain, we introduced an extension of this domain with increased difficulty by introducing new partial observability in the perception of the controlled agent.

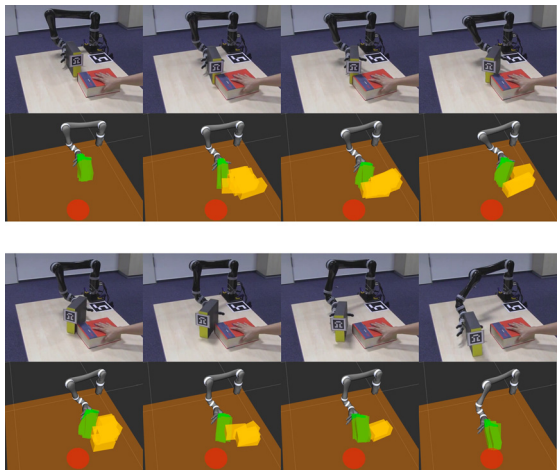


Fig. 18. Proposed approach w/ sampling: The robot arm performs a pushing task. The initial belief does not consider the static obstacle (red book support by a human hand) in the real scene that blocks the direct path to the goal region. After making contact with the obstacle at step 2, our method successfully generates suitable particles to update belief and represent the obstacle. POMCP is then able to choose suitable actions in order to avoid the obstacle and reach the goal.

Table 6
Hyper parameters for probabilistic distributions.

Target	Value
$\boldsymbol{x}_{pos}, \boldsymbol{x}^{obs}_{pos}$	μ, σ, a, b: Defined based on workspace of the robot
$\boldsymbol{x}_{ori}, \boldsymbol{x}^{obs}_{ori}$	$\mu = 0.0$, $\sigma = 0.15\pi$, $a = -0.45\pi$, $b = 0.45\pi$
m, m^{obs}	$\alpha = 1.5$, $\beta = 1.0$
μ_c	$\mu = 0.5$, $\sigma = 0.5$, $a = 0.0$, $b = 1.0$
$\boldsymbol{p}_{size}$, $\boldsymbol{p}^{obs}_{size}$	$\mu = 0.05$, $\sigma = 0.15$, $a = 0.025$, $b = 0.20$
$\boldsymbol{x}_{com}$	$\mu = 0$, $\sigma = 0.1$, $a = -\boldsymbol{p}_{size}$, $b = \boldsymbol{p}_{size}$
d_{type}	$(p_{box}, p_{cylinder}, p_{sphere}) = (0.34, 0.33, 0.33)$
d^{obs}_{type}	$(p_{box}, p_{cylinder}, p_{sphere}, p_{none}) = (0.25, 0.25, 0.25, 0.25)$

Experiments in the object manipulation domain were performed both in simulation and on a real robot. The robot had to perform object pushing tasks despite unknown object type and properties. In addition, unknown obstacles provided dynamics with discontinuities. The proposed approach showed good performance and could improve over the state-of-the-art planner. It showed better scalability with respect to increasing task difficulty. Standard POMCP would suffer from increasing particle deprivation because of the discontinuities in the dynamics. The proposed RNN based particle reinvigoration was able to overcome this problem.

Overall, we showed that our approach is able to adapt to mismatches between assumed and real situations while completing tasks. Model adaptation has been successfully used for a variety of models ranging from agent behavior to object and environment models. The approach handles dynamics with discontinuities well as the adapted model parameters are directly used in the physics engine as forward model.

It has to be noted that the performance is expected to decrease if an encountered situation is too far away from the data distributions used for pre-training the RNN. In such cases, our approach is expected to be capable of finding the best fit values within the available set of physical parameters. For example, in the object manipulation setting, unknown shapes will be approximated with the best fitting known shape and in many cases could still enable successful task completion.

Declaration of competing interest

There are no competing interests.

Appendix A. Hyper parameters for randomization

Simulations for the pushing task were performed in randomized worlds. For this, we sample the full physical parameters and initial states by using univariate probabilistic distributions. Table 6 provides some additional parameters to complement Table 3.

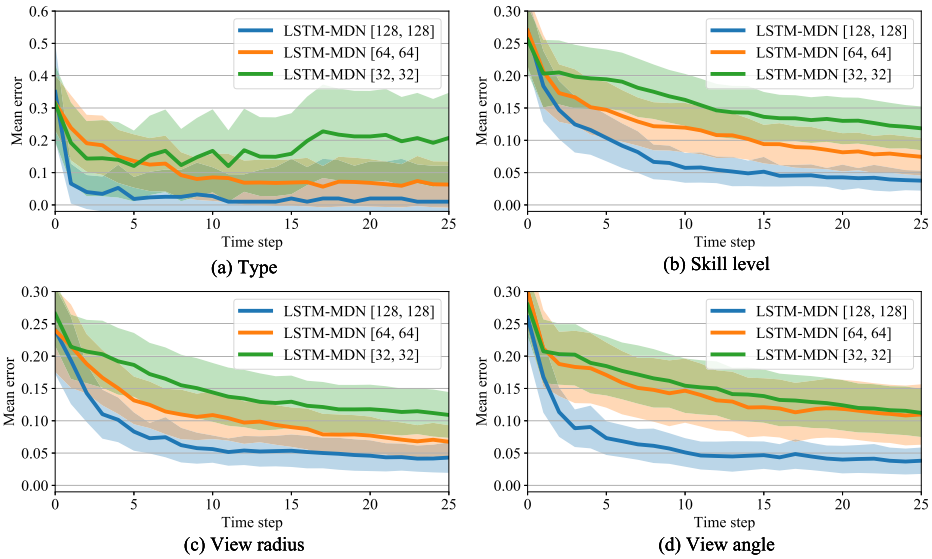


Fig. 19. Effect of different numbers of nodes in 2 hidden layers. Less numbers of nodes reduce accuracy of prediction of all parameters.

Appendix B. Hyper parameters effects on learning

Related to Fig. 7, we investigated an effect of different numbers of nodes (Fig. 19). All of the architectures have two LSTM layers. Numbers of nodes for each layer are differed as 128, 64, and 32. The figure shows that Increased numbers of nodes help to capture complex mapping between inputs (sequence of observations) and outputs (states and physical parameters).

References

[1] S. Albrecht, S. Ramamoorthy, A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems, in: Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-13), International Foundation for Autonomous Agents and Multiagent Systems, St. Paul, Minnesota, USA, 2013, pp. 1155–1156.
[2] S. Albrecht, P. Stone, Reasoning about hypothetical agent behaviours and their parameters, in: Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-17), International Foundation for Autonomous Agents and Multiagent Systems, São Paulo, Brazil, 2017, pp. 547–555.
[3] S. Albrecht, P. Stone, Autonomous agents modelling other agents: a comprehensive survey and open problems, Artif. Intell. 258 (2018) 66–95, https://doi.org/10.1016/j.artint.2018.01.002.
[4] K.J. Aström, B. Wittenmark, Adaptive Control, Dover Publications, 2008.
[5] S. Barrett, P. Stone, Cooperating with unknown teammates in complex domains: a robot soccer case study of ad hoc teamwork, in: Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015, pp. 2010–2016.
[6] C.M. Bishop, Mixture Density Networks, Technical Report, Aston University, 1994.
[7] G.M.B. Chaslot, M.H. Winands, H.J. van Den Herik, Parallel Monte-Carlo tree search, in: International Conference on Computers and Games, Springer, 2008, pp. 60–71.
[8] Y. Chebotar, K. Hausman, M. Zhang, G. Sukhatme, S. Schaal, S. Levine, Combining model-based and model-free updates for trajectory-centric reinforcement learning, arXiv preprint arXiv:1703.03078, 2017.
[9] E. Coumans, Y. Bai, Pybullet, a python module for physics simulation for games, robotics and machine learning, http://pybullet.org, 2016–2018.
[10] M. Deisenroth, C.E. Rasmussen, PILCO: a model-based and data-efficient approach to policy search, in: Proceedings of the 28th International Conference on Machine Learning (ICML-11), 2011, pp. 465–472.
[11] P. Doshi, Y. Zeng, Improved approximation of interactive dynamic influence diagrams using discriminative model updates, in: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems-volume 2, International Foundation for Autonomous Agents and Multiagent Systems, 2009, pp. 907–914.
[12] C. Finn, P. Abbeel, S. Levine, Model-agnostic meta-learning for fast adaptation of deep networks, arXiv preprint arXiv:1703.03400, 2017.
[13] C. Finn, S. Levine, Deep visual foresight for planning robot motion, in: 2017 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2017, pp. 2786–2793.
[14] J. Fu, S. Levine, P. Abbeel, One-shot learning of manipulation skills with online dynamics adaptation and neural network priors, in: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2016, pp. 4019–4026.
[15] P.J. Gmytrasiewicz, P. Doshi, A framework for sequential planning in multi-agent settings, J. Artif. Intell. Res. 24 (2005) 49–79.
[16] N.J. Gordon, D.J. Salmond, A.F. Smith, Novel approach to nonlinear/non-Gaussian Bayesian state estimation, in: IEE Proceedings F-Radar and Signal Processing, IET, 1993, pp. 107–113.
[17] A. Graves, Generating sequences with recurrent neural networks, arXiv preprint arXiv:1308.0850, 2013.
[18] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (1968) 100–107.
[19] H. He, J. Boyd-Graber, K. Kwok, H. Daumé III, Opponent modeling in deep reinforcement learning, in: International Conference on Machine Learning, 2016, pp. 1804–1813.
[20] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780.
[21] L.P. Kaelbling, M.L. Littman, A.R. Cassandra, Planning and acting in partially observable stochastic domains, Artif. Intell. 101 (1998) 99–134.
[22] L. Kocsis, C. Szepesvári, Bandit based Monte-Carlo planning, in: European Conference on Machine Learning (ECML), 2006, pp. 282–293.

[23] H. Kurniawati, D. Hsu, W.S. Lee, SARSOP: efficient point-based POMDP planning by approximating optimally reachable belief spaces, in: Robotics: Science and Systems (RSS), 2008.
[24] I.D. Landau, R. Lozano, M. M'Saad, et al., Adaptive Control, vol. 51, Springer, New York, 1998.
[25] S. Levine, P. Abbeel, Learning neural network policies with guided policy search under unknown dynamics, in: Advances in Neural Information Processing Systems, 2014, pp. 1071–1079.
[26] A.J. Lockett, C.L. Chen, R. Miikkulainen, Evolving explicit opponent models in game playing, in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, ACM, 2007, pp. 2106–2113.
[27] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533.
[28] C.H. Papadimitriou, J.N. Tsitsiklis, The complexity of Markov decision processes, Math. Oper. Res. 12 (1987) 441–450.
[29] J. Peters, S. Schaal, Reinforcement learning of motor skills with policy gradients, Neural Netw. 21 (2008) 682–697.
[30] R. Platt Jr, R. Tedrake, L.P. Kaelbling, T. Lozano-Perez, Belief space planning assuming maximum likelihood observations, Robotics: Science and Systems (RSS) (2010).
[31] A. Punjani, P. Abbeel, Deep learning helicopter dynamics models, in: 2015 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2015, pp. 3223–3230.
[32] R. Rahmatizadeh, P. Abolghasemi, A. Behal, L. Bölöni, From virtual demonstration to real-world manipulation using LSTM and MDN, arXiv preprint arXiv:1603.03833, 2016.
[33] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (1986) 533.
[34] J. Scholz, M. Levihn, C. Isbell, D. Wingate, A physics-based model prior for object-oriented MDPs, in: International Conference on Machine Learning, 2014, pp. 1089–1097.
[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347, 2017.
[36] D. Silver, J. Veness, Monte-Carlo planning in large POMDPs, in: Advances in Neural Information Processing Systems (NIPS), 2010, pp. 2164–2172.
[37] R.D. Smallwood, E.J. Sondik, The optimal control of partially observable Markov processes over a finite horizon, Oper. Res. 21 (1973) 1071–1088.
[38] A. Somani, N. Ye, D. Hsu, W.S. Lee, DESPOT: online POMDP planning with regularization, in: Advances in Neural Information Processing Systems, 2013, pp. 1772–1780.
[39] P. Stone, G.A. Kaminka, S. Kraus, J.S. Rosenschein, et al., Ad hoc autonomous agent teams: collaboration without pre-coordination, in: AAAI, 2010, p. 6.
[40] J. Sturm, C. Stachniss, W. Burgard, A probabilistic framework for learning kinematic models of articulated objects, J. Artif. Intell. Res. 41 (2011) 477–526.
[41] F. Wu, S. Zilberstein, X. Chen, Online planning for ad hoc autonomous agent teams, in: Twenty-Second International Joint Conference on Artificial Intelligence, 2011.