Preference Learning on the Execution of Collaborative Human-Robot Tasks

Thibaut Munzer[1], Marc Toussaint[2] and Manuel Lopes[3]

***Abstract*— We present a novel method to learn human preferences during, and for, the execution of concurrent joint human-robot tasks. We consider tasks realized by a team of a human operator and a robot helper that should adapt to the human's task execution preferences. Different human operators can have different abilities, experiences, and personal preferences, so that a particular allocation of activities in the team is preferred over another.**

We cast the behavior of concurrent multi-agent cooperation as a semi markov decision process and show how to model and learn human preferences over the team behavior. After proposing two different interactive learning algorithms, we evaluate them and show that the system can effectively learn and adapt to human preferences.

I. Introduction

Robotic technology is improving to the level that now we can have a close human-robot collaboration. This opens many new *cobotic* applications where people and robots work together in teams. In this context, robots can have simple repetitive behaviors and allow the user to adapt to them, or having reasoning capabilities that makes them true team members. A true collaboration involves shared decision making, mutual support, and easy ways to interact and express preferences.

In this work we want to explore such intuitive collaborations where the robot models the concurrent aspects of the shared task, and not just executes the task jointly with the user but also executes it in the way the user prefers. Such preferences are learned in an intuitive interaction loop.

We take this approach because to improve comfort, efficiency, acceptance, and reducing physical fatigue it should be the robot to learn and adapt to human preferences rather than the other way around. Such preference are learned during task realization to avoid periods of task programming facilitating task switching.

Recent works have considered preference learning and learning from imitation, but mostly in the context of single-agent tasks. In this paper we propose different ways to learn the preferences over the execution of a collaborative task with concurrent actions. Our contributions include the following: i) a formalism to model team behavior using a Relational Activity Processes (RAPs) [1]; ii) a formalism to represent preferences with standard MDP tools and iii) two different algorithms to learn preferences interactively. We show results in simulation and with a real robot.

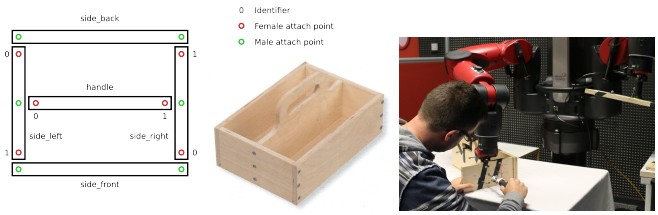

Fig. 1. The toolbox domain. A box can be assembled in a easier and efficient way if a collaborator robot helps by providing the new parts and by holding parts in place to make screwing easier for the user.

II. Related work

Research on learning from demonstration has included: i) low-level learning, where the goal is to learn a mapping from sensor output to motor command, e.g. learning motor policies [2] or navigation [3], [4]; ii) symbolic learning, where the goal being is to learn a policy, a mapping from a symbolic space state to the space of actions [5] or learning rewards [6] from demonstration.

In most of those examples, there is a clear separation between the learning and the execution phase. This has several drawbacks as the number of demonstrations might be larger than needed, and might not even cover critical aspects of the task. To address such problems interactive scenarios where both phases are merged have been proposed. In [2], the robot only makes queries when, in a given state, the confidence on the actions passes a given threshold. Alternatively, in [7], the system request information about relevant states to learn a good reward representing the task. Other approach provide a smooth transition between the phases, a first phase of teleoperation where the policy [8], or the preference [9], is learned and, at any time, the user can resume teleoperating to provide corrections.

A new trend of interactive learning systems is to rely on weak feedback to handle situations where optimal feedback is impossible or costly to produce by the human teacher. In particular, [10], [4] relies on local improvements of the current policy while [11] asks for ranking between two or more policies.

Another line of research considers not just learning individual tasks but also how to learn collaborative tasks. Several works have shown that learning the preferences of the human teammate has a positive impact on both cooperation and engagement of the user [12], [13].

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 and by the EU FP7-ICT project 3rdHand under grant agreement no 610878

[1]Thibaut Munzer is with Inria, France

[2]Marc Toussaint is with USTT, Germany

[3]Manuel Lopes is with INESC-ID, Instituto Superior Técnico, Portugal manuel.lopes@tecnico.ulisboa.pt

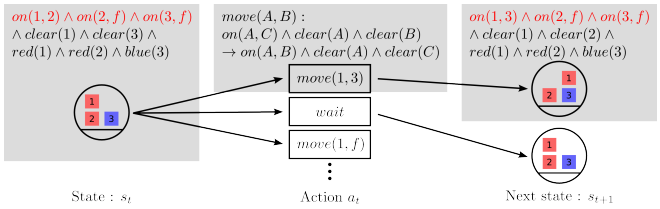

Fig. 2. Sketch of a Relational MDP representation of a blocksworld domain. The state is defined as a set of predicates, upon performing a relational action, the state, i.e. the true predicates change.

If the preferences of the user are known then planning methods can be used to anticipate the needs of the user [14]. [15] show how to learn different profiles of user's preferences by clustering trajectories containing multiple types of execution. The preferences of the user are modeled as a hidden state of a POMDP allowing to adjust to the particular type of user in real time.

The concept of cross-training is explored in [16] where robot and the user simultaneously adapt to each other by switching roles. The robot learns directly a policy that better adapts to the user preferences. This improves team efficiency and acceptance metrics.

Our work is different from the previous research because we consider: i) how to learn interactively preferences in a collaborative setting; and ii) we learn preferences in a high-level relational formalism with concurrent actions.

III. Team behavior modeling with RAPs

In this section we present the underlying formalism to model team behavior. We rely on a recently introduced general formalism for relational MDPs with concurrent actions [1], and show how to apply it to team behavior.

A. Relational MDPs

Markov Decision Processes (MDPs) are widely used to represent the decision process of an agent that can act in an environment. An MDP is a quadruplet (S, A, R, T) with S the space of states, A the set of actions, R some reward function that the agent aims to maximize, and T the transition probability function that describes the dynamics of the world.

Relational MDPs generalize MDPs for high-level representations [17]. Solutions to the planning and learning problems can be found in the literature [18], [19]. Under this representation, the state of the environment is defined as the set of predicates (logical formulas) that are true. Actions, and the transition function, are represented as a set of rules that contain 3 parts, i) the action and its argument (action are predicates); ii) the states where such action can be applied, aka the context; and iii) the outcome or the final state.

The figure 2 depicts how a Relational MDP may represent a domain where one can move blocks over each other.

B. Relational Activity Processes

For most formalisms for markov decision processes, only one action can be chosen at a given time. But in most cooperative tasks two agents perform actions at the same time. Other formalisms allow such flexibility but are not markovian and so all the already known algorithms for task learning cannot be used.

A Relational Activity Processes (RAP), introduced in [1], is a way to model concurrent cooperation of multiple agents while being at the same time a markov process. Roughly, RAPs define a sequential Markovian decision process where decisions are about the initiation or termination of activities, and activities run concurrently with random durations.

Under the RAP formalism we consider a relational domain with a set of predicates. For a given set of objects or agents, the state is the conjunction of all true grounded predicates. Under RAP we augment the state with some predicates representing running activities. These predicates include a special real argument which parameterizes the expected time-to-go of the activity. The decision set contains the initiations of all activities that can be started in the current state and a special *Wait* decision after which the real time advances until the next activity ends and applies its termination effects change the state. In contrast, for the initiation and termination of activities real time does not advance.

C. Decentralized Team Decision Making

Using the RAP formalism we can model teams of cooperating agents, where all agents are embedded in the same semi-MDP and the decision space is the *joint* space of human decisions and robot decisions, $\mathcal{D}$. Given a reward and using planning methods, for example Value Iteration, we can compute an optimal Q-function over the next decision $d \in \mathcal{D}$ in a given RAP state s. It provides values for decisions across agents. If there was a single central decision maker, it could read out the $\arg\max_d Q(d, s)$ and transmit the decision d to the agent it concerns. However, in real human-robot collaboration, without such a central decision maker, the readout and interpretation of this Q-function is non-trivial. With two decision makers, both might want to start an activity at the same time.

At team level, we transform this concurrent decision making into a sequential process. If both agents decide not to start a new activity the RAP wait decision is chosen. On the other hand, if the robot agent decides to engage an activity this is picked and transitions are applied. Human decisions can only be taken into account by the model when the robot decides not to start a new activity. This might come as a limitation but because all decisions except wait are instantaneous (as it is equivalent to start an activity), in practice, the human agent can engage a new activity at any time.

In the human-robot cooperation case we assume that the joint decision space space decomposes as $\mathcal{D} = \mathcal{D}_R \cup \mathcal{D}_H$ and $\mathcal{D}_R \cap \mathcal{D}_H = \{wait\}$, with $\mathcal{D}_R$ the robot's and $\mathcal{D}_H$ the human's decision space.

Given the robot has a representation of the shared task as a Q-function, we propose the following procedure for the robot to decide on its own activities. If $\max_{d \in \mathcal{D}_R} Q(d, s) < \max_{d \in \mathcal{D}_H} Q(d, s)$, that is robot decisions have strictly less

value than human decisions, the robot does not start an activity and lets the human act. Otherwise, the robot samples uniformly from the set of optimal robot decisions $\subseteq \mathcal{D}_R$.

IV. Interactive Preference learning

In the previous section we showed a general formalism to describe cooperative concurrent tasks. Now we present our interactive preference learning formalism. We show how we can model preferences in a cooperative MDP and then how can we learn them from a database in a batch function and then how to learn them interactively.

For the batch process, we assume having access to a dataset of state-action pairs $D = [(state_i, decision_i)]_{i=0}^{N}$ of behavior from the team that reveals the human preferences.

A. Modeling Preferences

Given a task with different ways to solve it, i.e. different optimal paths in the RAP. Preferences are defined as the preferred subset of these paths. The task can be seen as prior knowledge whereas preference is the learned behavior. In the extreme case where no prior knowledge is available this problem is reduced to learning from demonstration in an interactive setting.

More formally, under an MDP the task is defined by the reward, R_{task}. Using Value Iteration, we can compute the optimal quality function Q^*_{task}. We can then represent the behavior of the team taking into account human preferences as another quality function $Q^*_{full} = Q^*_{task} + Q_p$. Where Q_p is a shaping function of the task optimal quality function such that Q^*_{full} maximizes task and human preference.

In the following we will present two different methods to learn preferences.

B. Regression Based Preference Learning (RBPL)

The first method follows a typical approach of policy matching where we learn a function that directly represents the policy. We can see Q_{full} as the policy because we can directly compute a policy from a Q either by argmax or using some soft version like a Boltzmann policy.

With the decomposition we performed between task and preferences we can match the observed policy using the following process. For every state in D we want to reduce the quality of decisions not taken by the expert human-robot team, and increase the quality of decision taken.

For this we augment the dataset D and create a dataset D_f where for every couple $(state, decision)$ in D, for every decision e in $\mathcal{D}$ we add $((state, e), 0)$ if $e = decision$ or $((state, e), -1)$ otherwise to D_f.

We have now cast the learning problem as a regression problem. We can then rely on a relational regression tree such as the ones presented in [20] to learn the function Q_p from D_f. Such method have been shown to be successful in the context of learning individual policies in relational domains [6].

A relational tree follows a similar idea of typical decision trees but each node is a logical query on a given predicate. Several regularization methods can be used, a simple one is to limit the depth of the tree.

Algorithm 1 RBPL

```
procedure RBPL(Q*_task, D)
    D_f ← build_regression_dataset(D)
    Q_p ← learn_regression_tree(D)
    return Q*_task + Q_p
```

C. Gradient Based Preference Learning (GBPL)

The second idea follows a gradient based approach. In this case we do not have a set of parameters to optimize, and we rely on relational trees to learn functions. As such, we can not use classical gradient methods that work on parameters, but instead rely on functional gradient boosting where the gradient is computed on the space of functions. By defining a smooth mapping between policies and Q-functions (we use a boltzman mapping), it is possible to compute the values of functional gradient of the log-likelihood of the dataset D with respect to Q^*_{task} for every data point in D. Using relational regression tree, we can learn the functional gradient, f_0, that generalize gradient value to unseen data. We set $Q^*_{full} = Q^*_{task} + Q_0$. By iterating this process, as presented in Alg. 2, we can learn the function Q^*_{full} such that it maximizes the log-likelihood of the dataset D. This algorithm is analog to using the TBRIL algorithm [5] initialized with Q^*_{task}.

Algorithm 2 GBPL

```
procedure GBPL(Q*_task, D, nb_iter)
    Q*_full ← Q*_task
    for i ∈ [0, ..., nb_iter] do
        D_g ← compute_gradient(Q*_full, D)
        f_i ← learn_regression_tree(D_g)
        Q*_full ← Q*_full + f_i
    return Q*_full
```

D. Interactive setting

In the interactive setting, we use a similar approach. We initialize f to $f(s, d) = 0$. The robot takes decisions following the procedure describe above. After each decision taken (start human activity, start robot activity or wait) we update D. Two cases are possible:

- The decision is correct, we add $(state, decision)$ to the dataset, with $decision$ the robot decision
- the decision is incorrect, we add $(state, decision)$ to the dataset, with $decision$ a correct decision (given by the human as feedback).

Once the task is completed, we relearn f by one of the two ways above and the interactions continue to solve the task again, starting from a new state.

Because we modify the policy it can happen that the robot starts an activity that doesn't respect the preferences of the user or even that are not optimal with respect to the task. In some contexts, it might not be possible to recover from such error. This is the reason for another approach where we allow

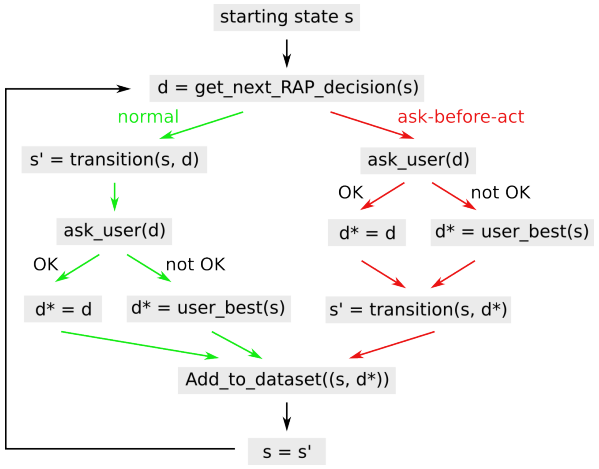

Fig. 3. Representation of the interactive process used by the robot to learn the preferences of its human teammate.

the robot to ask before doing an activity. We call this setup ask-before-act. Because this changes the MDP trajectories, it impacts the trajectory of learning. Both scenarios are shown in figure 3.

V. Simulations

A. Domains

We test our system in two domains: the blocksworld because it allows to easily change its dimension, and a more realistic cooperative human-robot assembling task for which we have a real robotic setup.

1) Concurrent blocksworld: This domain extends the standard blocksworld by allowing two activities to be executed at the same time. Blocks can be put on top of each other or on a surface (called floor) by a robot and by a human. Unless otherwise specified we use 5 blocks (2 red and 3 blue blocks).

The domain is represented with the predicates: *on/2*, *clear/1*, *busy/1*, *in_hand/2*, *blue/1* and *red/1*. The activities are *pick(agent, block)* and *put(agent, block, block)*, both of them last one unit of time and both of them can be realized by either the robot or the human.

The goal of the task is to stack all blocks in one tower. Starting states are generated by first sampling the number of initial towers between 4 and 5 and then uniformly select one state that respects the total number of towers. We enforce that no activities are active in the start state.

Even in such a simple domain we can imagine different user preferences. For instance one user might have a preference for a color, or for building the tower by alternating the actions with the robot. We first consider that the user dislikes picking blue blocks. Under such preference the task must be solved in a way that minimizes the number of blue blocks picked by the human while keeping an optimal policy with respect to the task.

2) Cooperative toolbox: The cooperative toolbox domain is inspired by industrial tasks. In this domain, represented in Fig. 1, a robot must support a human in the assembly of a toolbox. The toolbox is constituted by five pieces: handle, side_left, side_right, side_front and side_back. The toolbox can be built in different ways, the side_left and side_right are interchangeable as well as side_front and side_back.

At the beginning of the task, all pieces are set on a location not accessible by the human. The robot has to realize consecutively two activities to put them in the human workspace : pick(piece) and give(piece). Once the human has the pieces in his workspace he can start a positioning activity to put them in a correct disposition for screwing. Simultaneously, the robot should hold one of the pieces in order to allow the human to screw them together. Hold activity is done with the right arm of the robot whereas pick and give are done with the left arm so the robot can do different activities at the same time (which can be naturally represented with the RAP formalism). This domain has $240,000$ states.

In this domain, the task is to build the toolbox. Again we can imagine different ways to assembly the box that can be preferred by different people. For instance positioning everything and then screw, or positioning and screw one piece at a time. For the simulations we consider that the user likes to have an uncluttered workspace and so be given new parts as late as possible.

B. Results

To evaluate the proposed methods, we use three metrics. First, the accumulated preference reward gathered that measures if the robot found a policy that allows the users to execute the task in their preferred way. Second we count the number of errors: an error is made when a non optimal activity with respect to the task and preferences is realized, to evaluate the cost of learning the policy interactively. And third the number of human feedback needed to learn the preferences, to measure the cognitive cost of teaching the preferences on the task. These metrics are divided by the number of times the task has been solved. We could evaluate the learning offline (run a number of simulation after each learning episodes) but as we are interested in designing an online system we focus our efforts on online performance. These measure allows us to verify the compromise between instruct more and the number of errors.

1) Comparison of RBPL and GBPL: In Fig. 4 we compare the results of the RBPL and GBPL approaches on both domains. We can see that both approaches perform well on the toolbox domain. After 10 episodes the mean number of errors is close to zero and the accumulated preferences reward is close to the optimal. In the blocksworld domain the GBPL approach performs better in the long term.

We explain the difference of performance on the two domains by the complexity of the function to be learned. In the blockworlds, the function must be more generic because of the different possible starting states. The f function has to generalize in order for the agent to perform well. In that case, the gradient approach is more adapted as it can learn more complex model. On the toolbox domain, however, the

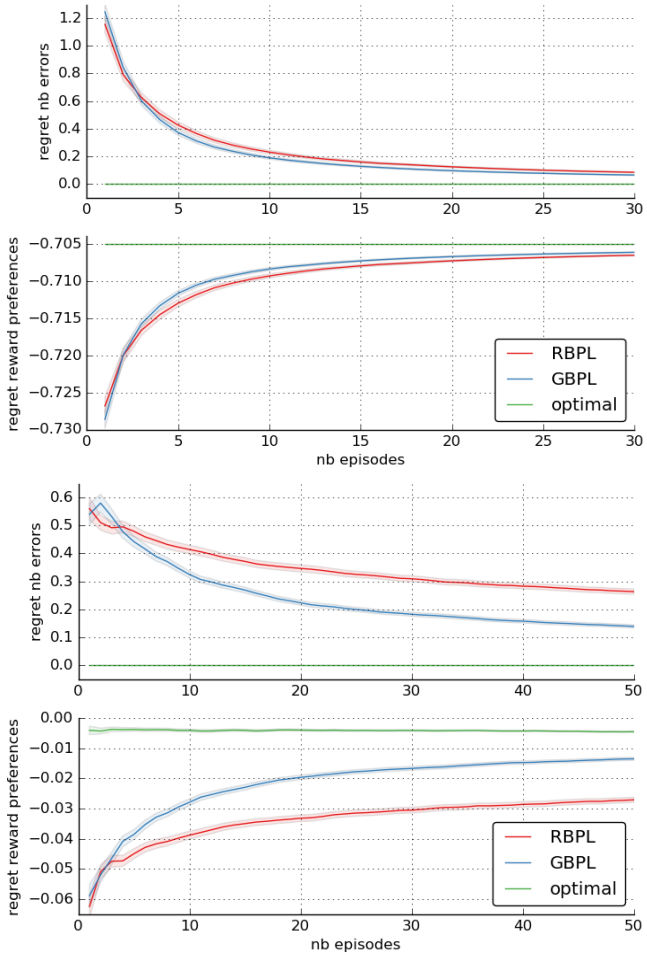

Fig. 4. Comparison of RBPL and GBPL on toolbox domain (top) and blocks domain (bottom). Both methods are able to learn to fulfill the task while respecting the user preferences. In this case, the number of feedback is equivalent to the number of error and is not displayed. Shaded area represent standard error of the mean.

starting state is always the same, and only local modifications of Q^*_{task} are needed in order learn correctly the preferences.

2) Ask-before-act: Figure 5 presents the number of feedback needed by the system to learn the user's preferences in the toolbox and blocksworld domains. We can see that for both domains ask-before-act leads to better results as the number of feedback decrease earlier. The system makes no mistakes and therefore the dataset D contain only states on the optimal RAP trajectories. This makes the learning task easier as f must be learned for a smaller part of its input space. For the blocksworld, as the learned function must generalize, the impact is less significant.

For the toolbox domain, the number of feedback is inferior at the beginning in the ask-before-act condition whereas it is the opposite for the blocksworld domain. In the first domain making a mistake with respect to preferences will lead to states where more mistakes can be made. In the second domain, it is the opposite, if the robot picks a red block, most of the time, no more mistakes can be made.

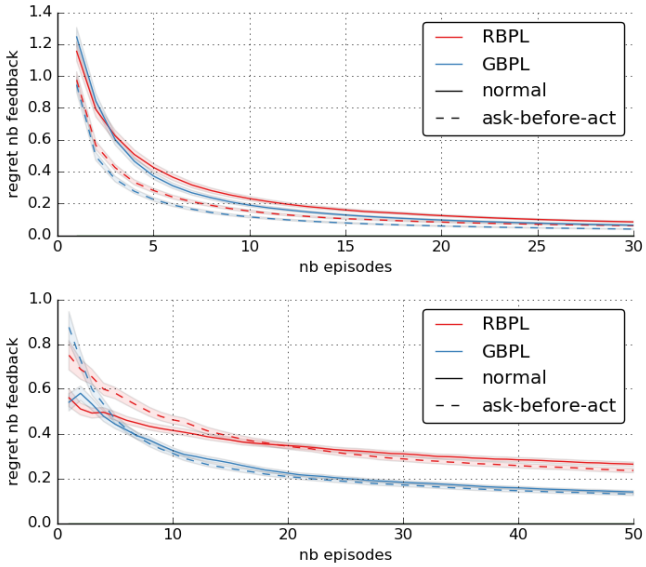

Fig. 5. Number of feedback for toolbox domain (top) and blocks domain (bottom). For ask-before-act the number of errors and the preference reward is equivalent to the optimal and is not displayed. Shaded area represent standard error of the mean.

3) Noise robustness: We now evaluate the impact of noise in the human feedback. We imagine that the human does not correct the robot every time. This creates a noisy learning scenario as wrong decisions will sometimes not be corrected by the human. It results in a dataset D where suboptimal decisions are present.

As figure 6 shows, even with noise equal to 0.4 (40% of wrong robot decisions that are not corrected) both approaches still manage to learn the human preferences. We observe that the number of feedback given in noisy setup becomes higher after only five episodes indicating that not always giving feedback is not a good strategy if one goal is to minimize the number of feedback given.

4) Transfer: We also evaluate the transfer capabilities of the learn policy. In the blocks world domain, a red block is added to the domain after 50 episodes.

Results reported in figure 7 indicate that when learned preferences are transfered, the learning is faster than when preferences are relearned from scratch. After only 10 episodes asymptotic performances are reached. We conclude that preferences can indeed be transfered from one domain to another.

VI. Robotic Experiment : Cooperative toolbox assembly

We now present an experiment of a joint human robot task realized with a user and the Baxter robot. This experiment uses the toolbox domain (figure 8 and consist in building the toolbox following a specific plan. In particular the user prefers having an uncluttered workspace while following this order for assembly:

1) attach handle and side_right
2) attach handle and side_left

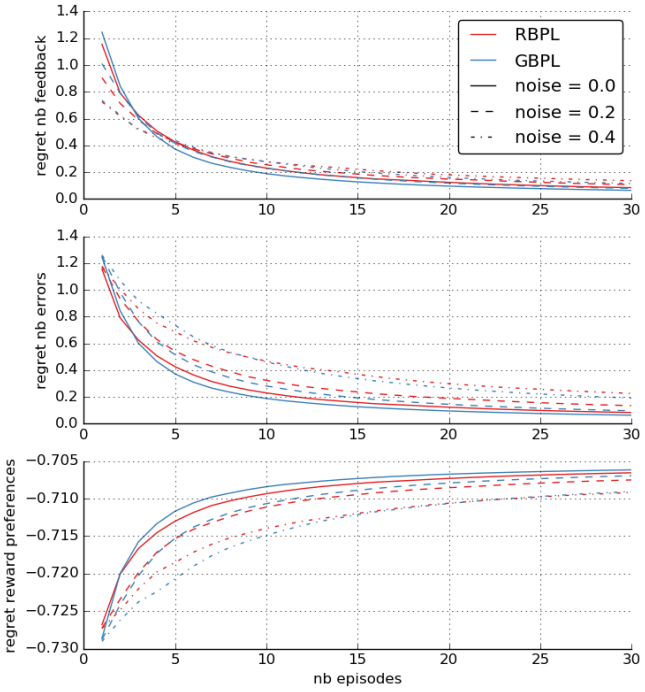

Fig. 6. Impact of the noise on the feedback in learning the task in the blocks domain. With noise up to 40% it is possible to learn the task. The standard error of the mean is not displayed for readability reasons.

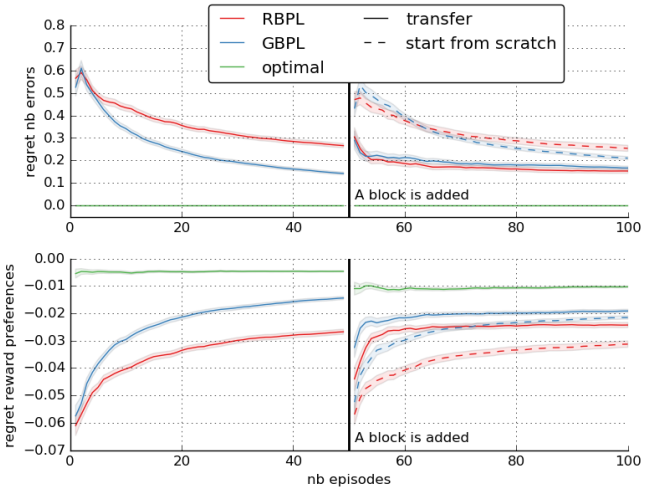

Fig. 7. Evaluation of the impact of changing the number of blocks during the task (at step 50). We can see that both approaches allow to transfer between domain sizes and learn faster than starting from scratch. In this case, the number of feedback is equivalent to the number of error and is not displayed. Shaded area represent standard error of the mean.

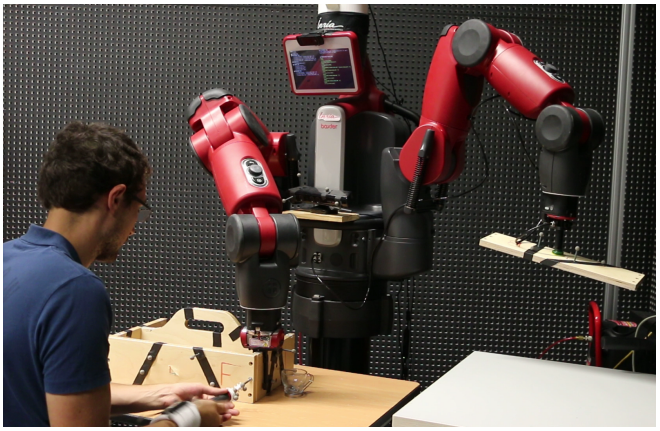

Fig. 8. Execution of a joint human robot while learning the user preferences.

3) attach side_right and side_front
4) attach side_left and side_front
5) attach side_left and side_back
6) attach side_right and side_back

During the first three episodes the starting state stays the same with all the pieces not in the human workspace. For the fourth episodes the side_front starts already in the human workspace.

The perception system relies on Optitrack cameras for object tracking. Based on this information, we compute the truth values of the domain predicates for any objects. The system is also programmed to recognize the different activities.

Using an algorithm on a real robot is always more challenging than in simulation. The list of additional difficulties includes : an imperfect perception system and a model not conform with the reality. The imperfect perception system leads to predicates wrongly detected as true as well as the other way around (for example, because of occlusion). Having an algorithm robust to noise helps coping with that. The mismatch of the model from the reality leads to making decision on states that are not predicted by the model. The presented algorithms allows learning what to do in those cases.

In this experiment, we use a hybrid strategy mixing the ask-before-act and the normal strategies. By applying the learning algorithm on different subsets of the dataset we can estimate the potential error (or confidence) of the predicted optimal decision. If the potential error is over a threshold, we use ask-before-act, otherwise the robot directly acts.

To compute the potential error, we learn 25 times on random subsets composed of 75% of the dataset. For every possible decision we compute the potential error as the mean over every learner of the difference between the predicted quality and the maximum predicted quality (by this learner). The predicted decision is the one with the lower associated potential error.

Based on the previous results, we use the GBPL algorithm as it has shown better performances. The results are presented

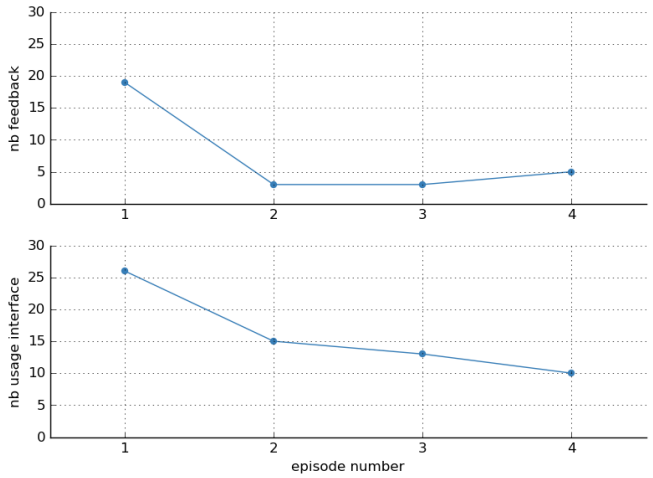

Fig. 9. Number of feedbacks (top) and number of uses of the interface (bottom) during four consecutive assemblies of the toolbox with a real robot. For the forth assembly, a different starting state is used. A decrease in the number of feedbacks shows the robot correctly learned the user preferences

in Fig. 9. We can see that the algorithm allows the system to significantly reduce the number of feedback after only one assembly. A feedback is given when the robot suggest a non preferred activity. So, a decrease in the number of feedback shows the robot correctly learned the user preferences. We also observe that the system is able to generalize to the new starting state in the forth assembly with a number of feedback of 5 as opposed to 19 for the first assembly. The bottom graph displays that the number of use of the interface decrease with the number of assembly. Which means that the robot is capable of correctly estimating its confidence. The video of the whole learning process can be found at https://vimeo.com/182913540.

VII. Conclusion

In this work we present an approach to learn preferences interactively during concurrent human-robot collaboration. In our setting the robot simultaneously executes the task and learns how the user prefers to execute it. Our main contribution is to consider such preference learning in an interactive and concurrent multi-agent action setting.

Our first contribution is to show how preferences can be efficiently encoded at the team behavior level using concurrent relational actions processes.

We then contributed with two preference learning methods where the team solves a shared task. These methods are prone to be used either in a batch or interactive setting. We showed results in a large dimensional domain, and a real robotic scenario, showing that these methods are able to learn how to execute the task while simultaneously learning which different ways of execution is preferred by the user.

We also showed how robust our approach is for intra-domain transfer and noise. We showed that we could change the dimension of the problem and still reuse parts of the information to learn fast. And, we showed that even is there is some noise in the feedback the system is able to learn.

References

[1] Marc Toussaint, Thibaut Munzer, Yoan Mollard, Li Yang Wu, Ngo Anh Vien, and Manuel Lopes. Relational activity processes for modeling concurrent cooperation. In *ICRA*, 2016.

[2] Sonia Chernova and Manuela Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34(1), 2009.

[3] W Bradley Knox, Peter Stone, and Cynthia Breazeal. Training a robot via human feedback: A case study. In *ICSR*, 2013.

[4] Ashesh Jain, Brian Wojcik, Thorsten Joachims, and Ashutosh Saxena. Learning trajectory preferences for manipulators via iterative improvement. In *NIPS*, 2013.

[5] Sriraam Natarajan, Saket Joshi, Prasad Tadepalli, Kristian Kersting, and Jude Shavlik. Imitation learning in relational domains: A functional-gradient boosting approach. In *IJCAI*, 2011.

[6] Thibaut Munzer, Bilal Piot, Matthieu Geist, Olivier Pietquin, and Manuel Lopes. Inverse reinforcement learning in relational domains. In *IJCAI*, 2015.

[7] Manuel Lopes, Francisco Melo, and Luis Montesano. Active learning for reward estimation in inverse reinforcement learning. In *ECML/PKDD*, 2009.

[8] Daniel H Grollman and Odest Chadwicke Jenkins. Dogged learning for robots. In *ICRA*, 2007.

[9] Martin Mason and Manuel Lopes. Robot self-initiative and personalization by learning through repeated interactions. In *HRI*, 2011.

[10] Pannaga Shivaswamy and Thorsten Joachims. Online structured prediction via coactive learning. *arXiv preprint arXiv:1205.4213*, 2012.

[11] Riad Akrour, Marc Schoenauer, and Michele Sebag. Preference-based policy learning. In *ECML/PKDD*. Springer, 2011.

[12] Min Kyung Lee, Jodi Forlizzi, Sara Kiesler, Paul Rybski, John Antanitis, and Sarun Savetsila. Personalization in hri: A longitudinal field experiment. In *HRI*, 2012.

[13] Noriaki Mitsunaga, Christian Smith, Takayuki Kanda, Hiroshi Ishiguro, and Norihiro Hagita. Adapting robot behavior for human–robot interaction. *Transactions on Robotics*, 24(4), 2008.

[14] Hema S. Koppula, Ashesh Jain, and Ashutosh Saxena. Anticipatory planning for human-robot teams. In *ISER*, 2016.

[15] Stefanos Nikolaidis, Keren Gu, Ramya Ramakrishnan, and Julie Shah. Efficient model learning for human-robot collaborative tasks. *arXiv preprint arXiv:1405.6341*, 2014.

[16] Stefanos Nikolaidis and Julie Shah. Human-robot cross-training: Computational formulation, modeling and evaluation of a human team training strategy. In *HRI*, 2013.

[17] Sašo Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine learning*, 43(1-2), 2001.

[18] Kristian Kersting, Martijn Van Otterlo, and Luc De Raedt. Bellman goes relational. In *ICML*, 2004.

[19] Tobias Lang and Marc Toussaint. Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research*, 39(1), 2010.

[20] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial intelligence*, 101(1), 1998.