Efficient Behavior Learning in Human-Robot Collaboration

Thibaut Munzer · Marc Toussaint · Manuel Lopes

Received: date / Accepted: date

Abstract We present a novel method for a robot to interactively learn, while executing, a joint human-robot. We consider collaborative tasks realized by a team of a human operator and a robot helper that adapts 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. Our main goal is to have the robot learn the task and the preferences of the user to provide a more efficient and acceptable joint task execution.

We cast concurrent multi-agent cooperation as a semi-Markov decision process and show how to model the team behavior and learn the expected robot behavior. We further propose an interactive learning framework and we evaluate it both in simulation and on a real robotic setup to show the system can effectively learn and adapt to human expectations.

Keywords Interactive Learning · Human-robot collaboration · Relational Learning

1 Introduction

Robots are still restricted to very controlled environments and mostly separated from humans. New technological developments have improved the safety of robots, making it possible to have robots and humans sharing the same space. For high volume production, it will still be more efficient to fully automatize the task. For low volume production, on the other hand, having a team composed of a human expert and an easily reconfigurable robot might be more efficient than full automation as the setup cost will be greatly reduced allowing for fast task switching. As a consequence, to be useful, robots should be able to help as soon as possible. In this context, robots should be able to learn how to assist a human operator. This process should be intuitive for the operator, requiring as little as possible knowledge about how the robot learns and acts. It is also important for this process to be time efficient, so that the learning phase is no longer than the actual task execution.

In this paper we propose an interactive learning system allowing a robot to learn to assist a human operator. We mean by interactive learning, fusing the training (creating a dataset of correct behavior) and execution (using the behavior learned from the dataset to solve the task) phases. It has several advantages: i) it exploits the current execution data to start act autonomously as soon as it is confident on the task, making the teaching process shorter tedious as the robot can act autonomously when it is sure about the correct action to execute; ii) the new experience and feedback from the user are used to fix the learned behavior if some parts are wrong or adapt if the expected behavior changes.

Another advantage of the interactive setting is to make it easier for a naive user to use the system. In many cases a deep understanding of the learning problem and learning algorithms is needed to select parameters such as the size of the training dataset, the amount of feedback, and the length of the training. Indeed, if
the size picked is too small the system will make mistakes with no possibility for improvement other than retraining the system with a bigger dataset. If the size is too big, the user will spend a lot of time recording useless demonstrations.

Recent works have considered interactive learning from imitation in robotics, but mostly in the context of single-agent tasks and without concurrent actions and high-level representations. In this paper, we propose a way to learn a collaborative task. The work presented here focuses on human-robot collaboration scenarios but could be applied more broadly, to any virtual agent that has to work with a human in a support fashion. Our contributions include the following: i) proposing and evaluating an interactive behavior learning framework for cooperative tasks with concurrent actions ii) the introduction of a potential error estimation mechanism for relational policy learning iii) the introduction of a formalism, based on relational semi-Markov decision processes, to model concurrent decision problem and iv) how to learn preferences with the proposed framework.

After reviewing related work in Sec. 2, we present the Relational Activity Process (RAP) framework and how to represent a cooperative task with it in Sec. 3. In Sec. 4, we present the interactive learning framework and the algorithms used. Lastly, Sec. 5 and Sec. 6 are the evaluation and the conclusion.

2 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 (Chernova and Veloso, 2009) or navigation (Knox et al, 2013; Jain et al, 2013); ii) symbolic learning, where the goal being is to learn a policy, a mapping from a symbolic state to the space of actions (Natarajan et al, 2011) or learning rewards (Munzer et al, 2015) 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 (Chernova and Veloso, 2009), the robot only makes queries when, in a given state, the confidence on the actions passes a given threshold. Alternatively, in (Lopes et al, 2009), the system request information about relevant states to learn a good reward representing the task. Other approaches provide a smooth transition between the phases, a first phase of teleoperation where the policy (Grollman and Jenkins, 2007), or the preference (Mason and Lopes, 2011), 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, (Shivaswamy and Joachims, 2012; Jain et al, 2013) relies on local improvements of the current policy while (Akour et al, 2011) 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 expected behavior from the human teammate has a positive impact on both cooperation and engagement of the user (Lee et al, 2012; Mitsunaga et al, 2008). If the preferences of the user are known then planning methods can be used to anticipate the needs of the user (Koppula et al, 2016). (Nikolaidis et al, 2014) 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 (Nikolaidis and Shah, 2013) where the 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 simultaneously: i) how to interactively learn behavior in a collaborative setting, ii) we learn behavior in a high-level relational formalism with concurrent actions, iii) to simultaneously do training and execution.

3 Team behavior modeling with RAPs

We model the concurrent task using Relational Action Processes (RAP) Toussaint et al (2016). The RAP framework relies on Relational MDPs, which model decision process where both state and actions use relational representations, and allows the concurrent execution of multiple actions that can also be initiated or terminated asynchronously.

3.1 Markov Decision Process

Markov Decision Processes (MDPs) are widely used to represent the decision process of an agent that acts in
an environment. An MDP is a quintuplet \((S, A, R, T, \gamma)\) with \(S\) the space of states, \(A\) the set of actions, \(R\) some reward function, \(T\) the transition probability function that describes the dynamics of the world, and \(\gamma\) the discount factor. The agent aims to maximize the sum of the discounted reward \(\sum_{t=0}^{\infty} \gamma^t r_t\).

A policy is the decision function of an agent. It gives the probability, given a state, to choose an action, \(\pi(s, a) = P(a|s)\).\(^1\) Under a policy \(\pi\) one can compute the so-called quality function:

\[
Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \sum_{a' \in A} \pi(s', a') Q^\pi(s', a')
\]

that computes the expected sum of discounted future rewards.

3.2 Relational MDP

Relational MDPs generalize MDPs for high-level representations (Dzéroski et al, 2001). This representation makes possible to model generic objects instead of specific instances of objects, allowing to generalize over objects, contexts, tasks and policies. Solutions to the planning and learning problems can be found in the literature (Kersting et al, 2004; Lang and Toussaint, 2010). Relational MDPs use a subset of first-order logic. We now define the main concepts in a simplified way:

- **Constant**: a constant is a symbol that refers to an object of the domain, usually a real world object but it can also be a number, a color, ...
- **Variable**: a variable is a symbol that can be substituted by a constant. We follow the classic prolog notation where identifier starting with a capital refer to variable and lowercase refers to constant
- **Term**: a term is either a constant or a variable.
- **Predicate**: a predicate is a symbol \(p/n\) where \(n\) is the arity of the predicate. It represents a relation between \(n\) constants or variables.
- **Atom**: an atom is an expression of the form \(p(a_1, ..., a_n)\) where \(p/n\) is a predicate and \(a_i\) are terms.
- **Formula**: a formula is a set of atoms and negated atoms.
- **Substitution**: a substitution is a mapping from variables to terms, noted \(\sigma = \{X_1 \rightarrow t_1, ..., X_n \rightarrow t_n\}\). Applying a substitution to a formula \(f\), noted \(f\sigma\) consist in replacing all occurrences of \(X_1, ..., X_n\) in \(f\) by, respectively, \(t_1, ..., t_n\).
- **Grounding**: a grounding is a substitution such that there are no variables in the mapped terms.

Under this representation, the state of the environment, given a set of predicates and a set of constants, is the set of all true grounded atoms.

For example, we can model, using a relational representation, an environment where three blocks are on the top of a table and can be stacked. There are four objects represented by the constants: \(b1, b2, b3\) and \(table\) and we use three predicates to represent the different states: \(on/2\), \(clear/1\) and \(block/1\). The state where \(b1\) is on top of \(b2\) and both \(b2\) and \(b3\) lie on the table will then be:

\[
\{on(b1, b2), on(b2, table), on(b3, table), clear(b1), clear(b3), block(b1), block(b2), block(b3)\}
\]

The state of the environment changes when relational actions are applied. In the blocksworld example, we can model a \(move(A, B, C)\) action that represents moving object \(A\) from the top of \(B\) to the top of \(C\). The action is abstract in the sense that it can be grounded to different objects. This action can represent moving \(b1\) from \(b2\) to \(table\), \(move(b1, b2, table)\) as well as moving \(b2\) from \(table\) to \(b1\), \(move(b2, table, b1)\).

Traditionally, in such domains the transition function is represented as a set of rules that contain 3 parts: the action, the context and the outcomes. There are different ways to represents it, we use the following one:

- the action and its argument (actions are atoms),
- the context, a formula that represents a precondition to executing the action,
- the outcome, a formula that represents the effect of executing the action,

and we represent them like this:

```
action:
  { context }
  { outcome }
```

The set of feasible grounded actions in a given state, \(D(s_t)\), is composed of all grounded abstract actions for which a rule, \(a : \{c\} \{o\}\), exists and there is a grounding \(\sigma\) such that the context is true in this state, \(c\sigma \subseteq s_t\). An abstract action can be present more than once in the decision set with different groundings.

If the agent decides to execute the action \(a\sigma\), the resulting state is obtained by applying the list of effects.

In our running example the transition function for the action \(move(A, B, C)\) will be represented with three rules:
We will continue to use our running example of the blocks world. However, to showcase the concurrent aspect, we introduce the presence of two agents and two activities will be used to realize the displacement of a block: pick(Agent, Block) and place(Agent, Block1, Block2). We also add two predicates, hand_free/1 and in_hand/2.

As for relational MDPs, the transition function is represented using rules. Given a set $\mathcal{A}$ of activity constants, for each activity $a \in \mathcal{A}$ there exist one or multiple initiation operators, encoded as rules of a Relational MDP. These initiators operators have a real value predicate $\text{go}(a) = \tau_a$ in the outcome set that represent the running activity. For instance:

We use the $\neg$ symbol to represent the negation.

The figure 1 illustrates a transition in the blocks world example.

![Sketch of an example of transition in the blocksworld domain](image)

**Fig. 1** Sketch of an example of transition in the blocksworld domain. The context is highlighted in blue, the positives outcomes are highlighted in green and the negatives ones in red. We favor red over blue. The rule used in the first action is the first one with the substitution $\{A \mapsto b_1, B \mapsto b_2, C \mapsto b_3\}$.

3.3 Relational Activity Processes

A RAP is a way to model concurrent cooperation of multiple agents as a relational semi-MDP (Toussaint et al, 2016). The formalism allows for multiple actions taken simultaneously and asynchronously.

In order to do so, it decomposes the action concept into two different concepts: decision and activity. Indeed, in the classical MDP formalism action encompass at the same the time the decision to start and the time to perform an action. By splitting it in two, and including the action being performed (called activity) in the Markovian state, RAP can represent a concurrent process. Roughly, RAPs define a sequential, semi-Markovian decision process where decisions are about the initiation of activities, and activities run concurrently with random durations.

For brevity, we only describe RAPs in the deterministic case. However, the interested reader can refer himself to (Toussaint et al, 2016).

We favor red over blue. The rule used in the first action is the first one with the substitution $\{A \mapsto b_1, B \mapsto b_2, C \mapsto b_3\}$.

3.3 Relational Activity Processes

A RAP is a way to model concurrent cooperation of multiple agents as a relational semi-MDP (Toussaint et al, 2016). The formalism allows for multiple actions taken simultaneously and asynchronously.

In order to do so, it decomposes the action concept into two different concepts: decision and activity. Indeed, in the classical MDP formalism action encompass at the same the time the decision to start and the time to perform an action. By splitting it in two, and including the action being performed (called activity) in the Markovian state, RAP can represent a concurrent process. Roughly, RAPs define a sequential, semi-Markovian decision process where decisions are about the initiation of activities, and activities run concurrently with random durations.

For brevity, we only describe RAPs in the deterministic case. However, the interested reader can refer himself to (Toussaint et al, 2016).
r1(X, A, B):
{ terminate(pick, X, A) on(A, B) }
{ ¬terminate(pick, X, A) in_hand(X, A) ¬clear(B) }

r2(X, A, B):
{ terminate(pick, X, A) ¬on(A, B) }
{ ¬terminate(pick, X, A) in_hand(X, A) }

r3(X, A, B):
{ terminate(put, X, A, B) block(B) }
{ ¬terminate(put, X, A, B) hand_free(A) on(A, B) }
{ ¬terminate(put, X, A, B) hand_free(A) on(A, table) }

Duration model: In the context of hierarchical RL and the standard Concurrent Action Model (Rohanimanesh and Mahadevan, 2005), steps of the sMDP correspond to the execution of an option, and the duration of the sMDP step is integer-valued, counting the steps of the underlying MDP. However, in general sMDPs the duration of one Markov step is real-valued, arbitrarily depending on \((s, d, s')\). In the concrete case of RAPs, we assume that initiation and termination decisions themselves have zero duration, while \(\tau\) is equal to \(\tau_{\text{min}}\) for the wait decision and therefore implicitly given by the \texttt{go} predicates in initiation operators.

Reward model: Rewards in RAPs are generally given as a relational mapping \((s, d, \tau, s') \rightarrow r\). In our applications, we encode this mapping as a relational tree as it is a compact and easily readable way to represent such mappings.

Optimality: Unrolling a policy generates an episode \((s_0, d_0, \tau_0, r_0, s_1, \ldots)\). We define the discounted return for an episode as

\[
R = \sum_{i=0}^{\infty} \gamma^i \beta(\tau_i) r_i, \quad \tau_i = \sum_{j=1}^{i} \tau_j, \beta(\tau) = \frac{1 - \gamma^\tau}{1 - \gamma} \quad (1)
\]

The \(\beta\) term weight each reward by the time taken by the transition while taking into account the impact of the discount factor.

3.4 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 (VI), 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 quality-function are non-trivial. With two decision makers, both might want to start an activity at the same time.

In the human-robot cooperation case, we assume that the joint decision space decomposes as \(\mathcal{D} = \mathcal{D}_R \cup \mathcal{D}_H\) and \(\mathcal{D}_R \cap \mathcal{D}_H = \{\text{wait}\}\), with \(\mathcal{D}_R\) the robot’s and \(\mathcal{D}_H\) the human 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 \(\max_{d \in \mathcal{D}_H} Q(d, s) < \max_{d \in \mathcal{D}_R} Q(d, s)\), that is robot decisions have strictly less value than optimal human decisions, the robot does not start an activity and lets the initiative to the human. Otherwise, the robot samples uniformly from the set of optimal robot decisions \(\subseteq \mathcal{D}_R\).

4 Interactive cooperative behavior learning

We now present the interactive learning framework. The basic idea is to have a system where the user can instruct the robot. But we make it also possible for the robot to act before being instructed when it is certain about what activity to execute. The underlying assumption is that, as the robot does the task again and again, it will become more and more certain about which activities to perform. To be certain about a decision we rely on potential error estimation (detailed in section 4.5).

4.1 The interactive framework

Fig. 2 presents the interaction flow. We incrementally build a dataset of expert behavior during task solving and after each episode (completion of the task) relearn the correct behavior using batch learning. During an episode, for each decision, based on the predicted decision and the potential error associated, we distinguish three cases: confident, ask-before-act and waiting-robot. If the predicted activity is a robot one and the estimated error is inferior to a threshold, called confident threshold, the robot acts and eventually gets feedback, if no feedback is given before the end of the activity, the activity is considered to be correct. On the other hand, if the estimated error is over the confident threshold, the robot starts by asking the user if the predicted decision is correct and use the user feedback to execute a correct activity. If the predicted activity is a human one or the decision is wait, the robot does nothing and gets feedback either by observing a human activity or by
getting a command to do an action. After each activity (start human activity, start robot activity or wait) D is updated.

Since our present goal is to produce an efficient learning procedure, we also added a memory system. If the robot recognizes the current state as part of D, it predicts the corresponding activity with an estimated error of 0. This also allows avoiding loops during the execution.

4.2 Learning cooperative behavior

We learn the behavior based on the dataset gathered during task solving. It is composed of tuples \((\text{state}_i, \text{decision}_i)\) where \(\text{decision}_i\) is a semi-MDP decision and can, therefore, be the activation of a human activity, the activation of a robot activity or the wait primitive.

We propose to use the TBRIL algorithm (Natarajan et al., 2011) as it is, to our knowledge, the only policy learning algorithm for relational knowledge. TBRIL works by finding a quality function such that seen actions are optimal for their associated state.

By first defining a smooth mapping between quality function and policy:

\[
\pi(s, a) = \frac{e^{-\beta q(s, a)}}{\sum_{b \in D(s)} e^{-\beta q(s, b)}},
\]

the problem becomes finding the function \(q^*\) that minimizes the negative log-likelihood of the dataset D. For-

\[
q^* = \arg \min(q(\pi(s, a)))
\]

TBRIL solves this problem using the Gradient Boosting (GB) algorithm (Friedman, 2001). GB it the transposition of the gradient descent algorithm in function space. Given a function space \(C \rightarrow \mathbb{R}\) and an objective function \(L\) to minimize, to apply gradient descent in function space, is to iteratively define \(f_{i+1} = f_i + \frac{\delta L}{\delta f_i}\) with \(f_0\) an initial solution (can be the null vector). More precisely, for any value of the input space, \(c \in C, f_{i+1}(c) = f_i(c) + \frac{\delta L}{\delta f_i}(c)\).

Gradient Boosting comes into play when it is impossible or not sensible to compute \(\frac{\delta L}{\delta f_i}(c)\) for every \(c \in C\), either because \(|C|\) is infinite or because there are \(c \in C\) for which values of \(f_i(c)\) as no influence on \(L(f_i)\). A weak regressor \(R_i\) is learned to predict/approximate the values of \(\frac{\delta L}{\delta f_i}(c)\) for any \(c \in C\). From a subset \(B \subset C\), a dataset is build \([c, \frac{\delta L}{\delta f_i}(c)]\) from which \(R_i\) is learned. \(f_{i+1}\) is then defined as \(f_{i+1} = f_i + R_i\).

In our case, the function space is \(S \times A \rightarrow \mathbb{R}\), the functor is \(L(Q) = \sum_{(s, a) \in D} -\log(\pi(s, a))\) and \(B\) is defined as \(B = \bigcup_{(s, a) \in D} \{(s, a)|a' \in D(s)\}\).

Because we are working with relational representations, we use as weak learner relational regression trees (Blockeel and De Raedt, 1998).

As a gradient algorithm, TBRIL needs a starting point. This starting point is a quality-function, \(Q_{prior}\), and can be set to 0. However, it can also be set to any quality function closer to the goal if prior knowledge is available, allowing to speed up learning. Alg. 1 sums up the algorithm.

\begin{algorithm}
\caption{TBRIL}
1: procedure TBRIL\((Q_{prior}, D, nb\_iter)\)
2: \(Q_{target} \leftarrow Q_{prior}\)
3: for \(i \in [0, ..., nb\_iter]\) do
4: \(D_g \leftarrow \text{compute\_gradient}(Q_{target}, D)\)
5: \(f_i \leftarrow \text{learn\_regression\_tree}(D_g)\)
6: \(Q_{target} \leftarrow Q_{target} + f_i\)
7: return \(Q_{target}\)
\end{algorithm}

4.3 Learning preferences

The previously defined algorithm can also be used to learn the user’s preferences when the general task is known beforehand.

Given a general 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 task learning.

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

By initializing the parameter \( Q^*_{\text{prior}} \) of TBRIL to \( Q^*_{\text{task}} \) we can use the same algorithm to learn preferences. The advantage of learning the preferences is to, at the same time, leverage prior knowledge to be efficient early on while being able to adapt to different users.

4.4 Protocol of interaction

At this stage, we introduce three feedback types:

- Validation: the predicted decision is correct
- Modification: the predicted decision is not correct and feedback is given before decision is executed
- Correction: the predicted decision is not correct and feedback is given after decision is executed

and two ways to get feedback:

- implicit: the system can recover the information by observing the scene
- explicit: the system receives the information by the direct intervention of the human

Based on the kind (robot/human/wait) of decision that is predicted and if it is correct, different feedback will be gathered. Table 1 presents these different feedback types. For example, if the predicted decision is correct, a validation feedback is be gathered. If the potential error is under the confident threshold, it will be done implicitly as the robot will act in confident mode. On the other hand, if the predicted decision is a robot one, the robot is confident and this prediction is wrong the feedback will be an explicit correction as the only way for the system to get feedback is from the user after the activity started.

An efficient learning procedure aims to minimize explicit feedback, as it requires attention from the user and can break the workflow of the task, as well as correction feedback, meaning a mistake have been made which can also break the workflow of the task. The challenge is due to these two objectives being opposed.

4.5 Estimating potential error

We now present how the potential error is estimated allowing acting differently when the system has enough information to be confident or when it should acquire more data before acting.

Because we are using relational trees to learn the expected behavior, we propose to estimate confidence using Query by Bagging (Mamitsuka, 1998).

In particular, we propose to learn \( N(= 50) \) models, each from a random subset, \( S_i \), of \( D \) (the optimal behavior dataset), i.e. \(|S_i| = 0.4 \times |D|\).

We need to consider a set of optimal decisions as in most cases there are more than one and the user might give different ones. We define the error of an optimal set of predicted decisions, a subset \( p \) of \( D(s) \), as the averaged minimum distance between a quality vector such that every decision in \( p \) is optimal, \( q_{pos} \), and the quality vectors predicted by the models.

\[
q^*_{pred} = \frac{1}{N} \sum_{n=0}^{N} \arg \min_{q_{pos} \in Q_p} |q_{pos} - q^*_{n_{pred}}|
\]

with \( q^*_{n_{pred}} = [Q^*_n(d,s)]_{d \in D(s)} \), the vector of quality values predicted by learner \( n \), and \( Q_p = \{\mathbb{R}^{|D(s)|}\}_{d \in p, d' \in D(s), t_d \geq t_d'} \), the set of quality vectors such that all decisions from \( p \) are optimal.

Given the state, for each possible set of decisions, the error is computed. The set with the least error is the final prediction. Sometimes more than one set shares the same error (for example, if only two decision, \( d_1 \) and \( d_2 \) are optimal for every learner, three sets will minimize the error: \( \{d_1\}, \{d_2\} \) and \( \{d_1, d_2\} \)) we pick the one with the higher cardinality.

For a given set of decisions, we cast the problem of finding the associated error as a quadratic optimization under constraints problem and use an off-the-shelf solver to solve it.

5 Evaluation

This section presents the different experiments we conducted to evaluate the framework. They are divided in simulation experiments and a robotic experiment. In simulations, we validate the approach and evaluate how it can handle different situations. In the robotic setup, we validate it can be used on a real world robot.

5.1 Simulation Experiments

The simulation experiments are conducted on two domains and evaluate different aspects: the impact of the
confident threshold, how the interactive approach performs compared to a batch one, how sensitive to noise the system is, how well it can transfer and what is the quality of the potential error. We use two metrics for most of these experiments. The cumulated number of explicit instructions and cumulated number of errors across different task executions. As explained earlier, both explicit instructions and errors break the workflow of the task and as such should be avoided.

5.1.1 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.

Concurrent blocksworld This domain extends the standard blocksworld by allowing two activities (pick and put) 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 uniformly draw one state such that the number of towers is 4 and no activities are running.

Cooperative toolbox The cooperative toolbox domain is inspired by industrial tasks. In this domain, represented in Fig. 3, 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 in 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 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.

5.1.2 Results

Impact of the confident threshold parameter Previously we introduce the confident threshold that controls the trade-off between the number of explicit instructions and the number of error of the system. Fig. 4 presents the cumulative number of explicit instructions and er-

<table>
<thead>
<tr>
<th>correct decision</th>
<th>A robot decision is predicted</th>
<th>Human decision or wait is predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>predicted</td>
<td>implicit validation</td>
<td>implicit validation</td>
</tr>
<tr>
<td>not predicted</td>
<td>explicit correction</td>
<td>explicit modification</td>
</tr>
<tr>
<td>(robot decision)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>not predicted</td>
<td>explicit correction</td>
<td>implicit modification</td>
</tr>
<tr>
<td>(human decision)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1 The different kind and ways to obtain feedback based on the predicted action and whether it is correct. The cases to avoid, explicit (blue) and correction (red) are highlighted.
errors for different value of \textit{confident threshold}. We can see that, for both domains, with high values, the number of explicit instructions is low but the number of error is high and the other way around for low values of the \textit{confident threshold}. It can indeed be used to control the trade-off between the number of explicit instructions and the number of error of the system. For application where errors are costly, it should be set to a low value whereas non-risky application should use a high value to increase learning speed.

\textbf{Incremental vs. Batch} A main claim of this article is that an interactive learning approach allows reducing the numbers of instructions and errors made by a system when compared to batch learning. Fig 5 displays the cumulative numbers of instructions and errors for interactive and batch after 20 task completions. Each approach has been run with different values for the parameter that controls for the trade-off between instructions and errors, \textit{confident threshold} for interactive and number of demonstrations (full solve of the task) for batch.

For evaluating the batch learning, the interactive process is used except for the following modifications, with \textit{nb\textsubscript{demo}} = \textit{n}:

- The policy is only learned once, at the end of the \textit{n}-th episode.
- Before the end of the \textit{n}-th episode, the robot will never use the confident mode.
- After the \textit{n}-th episode, the robot will never use the ask-before-ask mode.
- After the \textit{n}-th episode, the user will never give feedback to the robot.

The results show that for the blocksworld domain, for any values of \textit{nb\textsubscript{demo}}, there is a value of \textit{confident threshold} such that the interactive approach is better both in terms of explicit instructions and errors. This also true for the toolbox domain except for \textit{nb\textsubscript{demo}} = 10 where even a very low value of \textit{confident
threshold, the interactive approach makes more mistakes. We explain later why this is the case. We argue that in most case, interactive with $\text{threshold} = 0.0005$ will be preferable to batch with $\text{nb}_{\text{demo}} = 10$, the number of errors is only 0.3 more while the number of instructions goes down to 56 (from 115).

**Noise Sensitivity** In a real life scenario, the feedback might be noisy. For example, if the system is using speech recognition, some recognition error can occur or if the system is using a graphical interface, the user might miss-click. To evaluate the robustness of the proposed system to noise, we make the assumption that the communication to instruct the robot of the optimal action is noisy, for $\text{noise} = n\%$, in $n\%$ of the case a random decision is given in place of an optimal one. We can see in Fig. 6 that up to 20% noise the system is able to learn the policy efficiently. However, at 50% noise the performances decrease considerably.

**Preference learning** Fig. 8 evaluates the impact of leveraging prior knowledge about the task when such knowledge is available. In this experiment, we changed the task in both domains to add a preferences component. For the block domain, the user prefers to only handle red block. In the toolbox domain, the user prefers to get pieces one by one and as much as possible for the robot to have the arms in home position.

We can see that in both domains using the prior knowledge of the task allows learning faster while making no mistakes. **[it seems the reverse, the null initialization has zero errors.]**

**Potential Error Quality** We are interested in evaluating if the measure used to estimate the potential error is clearly predicts the future error. In Fig. 9 we plot the distribution of the potential error for correctly and
wrongly predicted decisions. We used the data of 45 runs of the interactive approach for 20 episodes with a confident threshold value of 0.01.

We can see that, for both domains, most of the correct decisions have a very low error (the y-axis uses a log scale). And, for the blocks world domain, the distribution of wrong decisions is clearly different. For the toolbox domain, however, the two distribution are more alike. It explains that, even for low values of the confident threshold, errors are made by the interactive approach.

5.2 Robotic implementation

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 3) and consists of a collaborative assembly of a toolbox. We run the system for three episodes, each following a different RAP trajectory. The first two start with all the pieces not in the human workspace but the handle piece is assembled differently (it is reversed). The third one starts with the side_front already in the human workspace.

The perception system relies on Optitrack cameras for object tracking and human activity recognition, both outside the scope of this paper. Based on this information, we compute the truth values of the domain predicates for all objects. The system is also programmed to recognize the different human 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 to cope with that. The mismatch of the model from the reality leads to making decision in states that are not predicted by the model. The presented algorithm allows learning what to do in those cases.
Fig. 10 Number of modification feedbacks (top) and number of uses of the interface (bottom) during three consecutive assemblies of the toolbox with a real robot. Because there were no error made by the robot, a decrease in the number of modification feedback shows the user needs to instruct less and so that the robot correctly learned the task.

The results are presented in Fig. 10. We set the confident threshold low enough such that no error are made by the system. We can see that the algorithm allows the system to significantly reduce the number of modification feedback after only one assembly. A modification feedback is given when the robot suggests a nonoptimal activity. So, a decrease in the number of modification feedback shows the robot correctly learned the task and is capable of generalizing. The bottom graph shows that the number of use of the interface decrease with the number of assemblies. Which means that the robot is capable of correctly estimating its confidence. The video of the whole learning process can be found at **[link]**.

6 Conclusion

In this work we present the first approach for a robot to interactively learn a support behavior during concurrent human-robot collaboration. In our setting, the robot simultaneously executes the task and learns what the user expect it to do. Our main contribution is to consider such behavior learning in an interactive and concurrent multi-agent setting.

We first detailed the RAP framework and how to use it to represent concurrent and cooperative task realization. Because RAP is constructed on an underlying relational semi-MDP model, one can use pre-existing policy learning algorithm such as TBRIL. After introducing an interactive behavior learning framework that mixes the usual training and exploiting phase, we presented an estimate of the potential error for a given activity prediction.

We have shown that using the proposed framework leads to the robot making fewer errors and requiring less explicit instruction than more traditional batch learning. We also evaluate how robust our approach is for intra-domain transfer and noise. We demonstrate that we could change the dimension of the problem and still reuse parts of the information to learn fast. We describe how to use the proposed framework to learn user’s preference during the execution of a collaborative task.

Lastly, we evaluated the implementation of this framework on a real robot and showed its viability.

We believe interactive learning is an important feature to allow naive users to teach behavior to a robot. Future works could include supporting a greater variety of feedback mechanism to make the interaction more natural.

Acknowledgements 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

References


