2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Active Learning of Manipulation Sequences

David Martínez[1], Guillem Alenyà[1], Pablo Jiménez[1], Carme Torras[1],
Jürgen Rossmann[2], Nils Wantia[2], Eren Erdal Aksoy[3], Simon Haller[4], Justus Piater[4]

***Abstract*— We describe a system allowing a robot to learn goal-directed manipulation sequences such as steps of an assembly task. Learning is based on a free mix of exploration and instruction by an external teacher, and may be active in the sense that the system tests actions to maximize learning progress and asks the teacher if needed. The main component is a symbolic planning engine that operates on learned rules, defined by actions and their pre- and postconditions. Learned by model-based reinforcement learning, rules are immediately available for planning. Thus, there are no distinct learning and application phases. We show how dynamic plans, replanned after every action if necessary, can be used for automatic execution of manipulation sequences, for monitoring of observed manipulation sequences, or a mix of the two, all while extending and refining the rule base on the fly. Quantitative results indicate fast convergence using few training examples, and highly effective teacher intervention at early stages of learning.**

I. Introduction

To this day, industrial robots almost always execute preprogrammed motor sequences, grasp and manipulate workpieces in fixed sequences, and follow preprogrammed trajectories. This requires a carefully controlled setup and costly programming. Modern applications such as manufacturing and service robotics require simple programming, e.g. by human demonstration, and should perform robustly in variable environments.

In this work, we describe a system that learns flexible manipulation sequences by a combination of observation and exploration. The centerpiece is a symbolic reasoning engine, referred to as the *Decision Maker* (DM), which represents manipulation sequences in terms of simple, object-level manipulations such as *open door*, *extract tray*, or *insert peg*. We assume that the robot is capable of executing these manipulations by virtue of appropriate object recognition, pose estimation, grasping and motion skills. Our innovation lies in the automatic, incremental acquisition of *rules* by the DM, allowing it to flexibly synthesize manipulation sequences that result in a desired outcome, to reactively replan in case of unexpected disturbances, and to monitor observed action sequences for correctness.

A rule is characterized by a manipulation, a set of preconditions, and a description of the outcome. These are incrementally learned and refined by observing a teacher performing manipulations while keeping track of the state of the workspace. In addition, learning can be active in that the DM can take the initiative and suggest or trigger next actions in the manipulation sequence. During learning, rules, preconditions and outcome descriptions will typically be incomplete, which will cause some suggested actions to fail. This yields the addition of new rules or the refinement of existing rules. Notably, our procedure does not require a deterministic environment, but explicitly takes uncertainty into account. The end result is a rule set allowing the DM to generate a variety of plans, all leading to the same goal state.

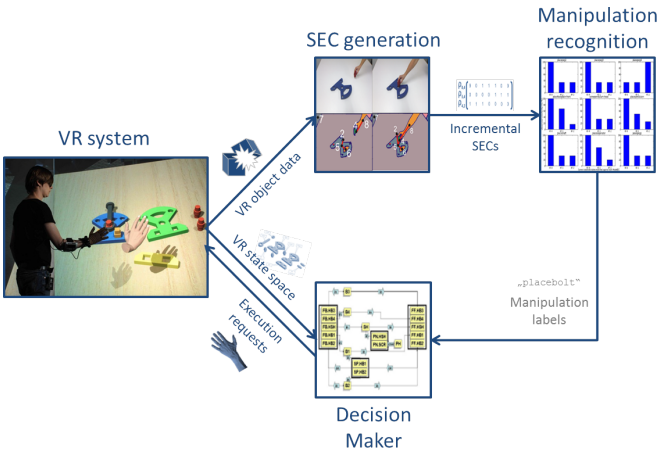

Fig. 1. System Overview

Figure 1 presents an overview of the system. The robotic actor and its environment are simulated in Virtual Reality (VR)[1]. The evolution of the manipulation sequence is extracted and represented in the form of so-called Semantic Event Chains [2], [3]. From these, the Manipulation-Recognition module extracts plausible actions in progress and recognizes completed actions. This information, plus additional state communicated by the VR system, is used by the Decision Maker to assess the state of the environment and to learn and refine rules.

II. Related Work

Many tasks in human environments can be modelled as the execution of a sequence of different actions. To introduce

[1]IRI (UPC-CSIC), Llorens i Artigas 4–6, 08028 Barcelona, Spain dmartinez,galenya,pjimenez,ctorras@iri.upc.edu
[2]MMI, RWTH Aachen, Ahornstr. 55, 52074 Aachen, Germany rossmann,Wantia@mmi.rwth-aachen.de
[3]Inst. Physics-3 & BCCN, University of Göttingen, Friedrich-Hund-Platz 1, 37077 Göttingen, Germany eaksoye@physik3.gwdg.de
[4]IIS, IFI, University of Innsbruck, Technikerstr. 21a, 6020 Innsbruck, Austria Simon.Haller,Justus.Piater@uibk.ac.at

[1]In this paper we focus on the interactive learning and planning. The system, as described, is also applicable to a real robotic setting, which is currently work in progress.

flexibility in the way robots are taught to perform such actions, different learning and cognitive frameworks have been introduced. The challenge is now to learn such actions in a manner that they can be exploited afterwards in a logic planning framework to produce plans that solve given tasks.

Most existing machine learning techniques are designed to work off-line, i.e. they require the samples for learning to be available in advance, typically use an iterative process that requires a significant amount of data and computation [17], and usually require a complete, manual specification of the problem in advance, like for example classical symbolic planning algorithms [5].

To become more flexible some authors have proposed learning frameworks to automatically learn different parts of the problem. Yang et al. [18] propose a framework where the preconditions of each action (under which circumstances an action can be executed) can be learnt using experience. However, it still requires the careful specification of the rest of the problem.

Recently, Sung et al. [13] propose to generate universal plans as sequences of primitive actions. Their selection takes into account environmental parameters and the current state in a parametrized form, and consequently tries to generalize the plans to unseen scenarios. This approach needs to define accurately the set of attributes for each of the involved objects, and more importantly, the effects and requirements of each action in advance.

Zhuo el al. [19] propose a method to refine a model with incomplete preconditions and effects of the actions using a MAX-SAT algorithm for learning. In their approach a set of valid partial plans in the unknown domain is required to find the solution.

Model-based reinforcement learning (RL) [7] allows learning with a limited amount of experiences. In the KWIK RL framework [10] a method to learn the probabilities associated to a set of given action effects using linear regression has been proposed [15], and also an extension to learn the action effects themselves [14]. To reduce the number of actions required in model-based RL, a compact representation of the model using relational rules can be used. Lang et al. proposed REX [9], a relational reinforcement learning algorithm that uses relational generalization with the well known R-max [4] and E^3 [6] algorithms to minimize the amount of exploration required.

In contrast to the above, we propose a complete framework for learning action rules *on-line*, starting with an empty rule base. The introduction of a teacher is needed in such a system where there is no previous knowledge. Teachers have already been used to supervise the learning process [16]. However, teacher time is usually very valuable. Therefore we give the system the ability to *request* the help of the teacher just whenever it is needed. In both cases, the introduction of a teacher improves the efficiency of the learning [1] because the number of learning iterations required to gather enough knowledge to complete the required task is decreased. We propose the REX-D algorithm, an extension of the REX algorithm with the inclusion of teacher demonstration requests.

III. From Signals to Symbols

Observed actions give rise to a continuous sensory stream, which has to be translated into a symbolic form suitable for the Decision Maker. This is the role of the "SEC generation" and "Manipulation recognition" modules described in this section.

In this work, we focus on reasoning about manipulations as represented by dynamic relations between objects. To this end, we here avoid the compounding challenges of observation and execution of real-world actions, and work within a Virtual-Reality environment. However, in principle, the VR system can be replaced by real-world sensing (e.g. using RGB-D cameras) and acting (robots and humans), while leaving the other modules largely unchanged. We address this in current work.

A. Semantic Event Chains

In the setup of Fig. 1, the VR system represents the environment as a dynamic state space. Each state includes identity, pose, and spatial information about objects and the hand present in the scene, and is updated dynamically during the manipulation.

The state information is first represented by a graph: nodes correspond to object centers (which may include a human hand), and edges indicate whether two objects touch each other or not. The touching relation is directly derived from the spatial information embedded in state descriptors. In this way, a manipulation sequence is described by a sequence of graphs.

Using an exact graph matching technique, the "SEC generation" module drops successive identical graphs (i.e., it deletes all graphs isomorphic to its predecessor). Thus, each remaining graph represents a "key frame" in the manipulation sequence, characterized by a change in the visible object set or in relations between objects. From these remaining graphs, we form a so-called *Semantic Event Chain* (SEC), which is a matrix where rows are possible pairwise object touching relations, and columns describe the scene configuration when a new graph has occurred. A SEC is an attractive descriptor for manipulation actions since it is invariant to the particular objects used, to the precise object poses observed, and to the actual trajectories followed. Moreover, it is robust to a considerable amount of clutter nodes and edges that are unrelated to the action of interest. All these aspects are allowed to vary, and still the same SEC is observed and captures the "essence of the action", as demonstrated with diverse sets of real actions in our earlier work [2], [3].

B. Manipulation Recognition

The manipulation actions considered by the Decision Maker are all representable by sequences of changing object-object relations. For example, *insert peg A into object B* involves the following sequence of exhaustive touching relations, as illustrated in Fig. 2:

1) Peg and Object touch the table. The Hand is visible.

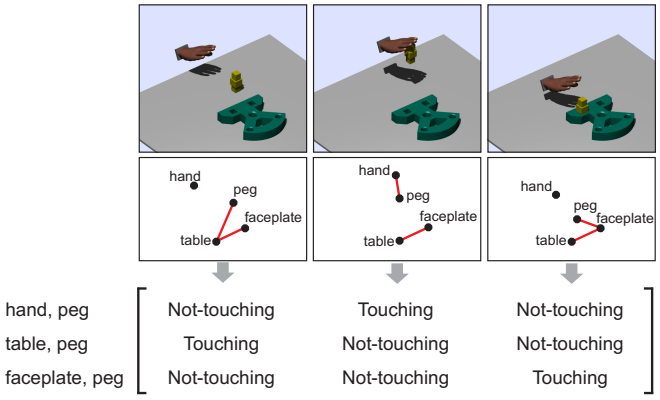

Fig. 2. Touching relations that define key-frame Scene Graphs (center row) that compose a Semantic Event Chain (the matrix at the bottom) representing a peg-in-hole manipulation.

2) The Hand touches the Peg, and the Object touches the table.
3) The Hand has released the Peg, which touches the Object, which in turn touches the table.

This corresponds to a characteristic triple of columns in a SEC. The role of the "Manipulation recognition" module is to parse an incoming, continuous SEC (without indication when a manipulation begins or ends) into a sequence of known (and unknown) manipulations in an online, incremental fashion, and to feed them to the Decision Maker for learning and online planning.

The "Manipulation recognition" module possesses a database of model SECs, each corresponding to one manipulation. Each model SEC contains three SEC columns (corresponding to keyframes) describing the atomic actions that have to appear in sequence for that specific manipulation. Online recognition of manipulations is done in two steps. The first step consists in matching the current SEC column, received from the SEC generation module, to all model SECs. The second step keeps track of the state of advancement of each model manipulation, and determines whether one or more manipulations have completed.

At the first step, the current SEC column, describing the change in spatial relationships between scene segments observed at the current time step t, is compared to all keyframes k of each model SEC m. A match score $s^{(t)}_{m,k}$ for each of these comparisons is calculated as the proportion of inter-segment relations (row entries of SEC columns) that match between the current SEC column and the model keyframe.

To implement the second step, a state counter $c^{(t)}_m$ is associated with each model SEC m that gives the highest keyframe k compatible with the sequence of observed atomic actions so far. Initially, all state counters are set to zero. Then, for each manipulation model m,

$$c^{(t)}_m = \underset{k \leq c^{(t-1)}_m + 1}{\operatorname{argmax}} \left\{ s^{(t)}_{m,k} = 1 \right\}. \quad (1)$$

This allows the state to advance keyframe by keyframe (corresponding to a correct progression without skipping atomic actions), or to be reset to an earlier state (corresponding to a recoverable step backwards in the sequence of atomic actions).

A manipulation m is *recognized* whenever $c^{(t)}_m = K$ for a model SEC consisting of K atomic actions. An *unknown state* is signaled if there is no match between the current SEC column and any model SEC, i.e., $s^{(t)}_{m,k} < 1$ for all m and all k. As soon as the first keyframe is matched thereafter, i.e., $s^{(t)}_{m,k} = 1$ for at least one (m, k), an *unknown manipulation* is reported for the preceding, unknown sequence of atomic actions.

IV. The Decision Maker

The centerpiece of our system is a decision making module (cf. Fig. 1) that allows a robot to iteratively learn to execute multi-goal, human-like tasks with the guidance of a human teacher. During the task execution, the robot incrementally learns new behaviours, becoming more autonomous up to the point that human guidance is no longer required. This framework is suitable to train robots to execute a wide spectrum of human-like tasks without the burden of coding the behaviours in advance or recoding the behaviours after any non-stationarities of the environments.

The Decision Maker receives experiences in the form of recognized manipulations (Sec. III-B), plus additional state information received directly from the VR system (Sec. V). It produces plans in terms of action sequences, which are used in two distinct modes of operation that differ in the use of these functions and in the roles of the human user:

- **Learning and execution:** The DM has to complete the tasks it is assigned, starting with no prior knowledge about the scenario. It has to request demonstrations, learn the model and plan action sequences until it completes the tasks.
- **Monitoring:** The decision maker verifies a task executed by an user. It recommends actions to be taken, and informs the user whether she is successfully approaching to the goal or not.

A. Learning and execution

The decision maker starts with no prior knowledge about the actions that can be executed and has to learn them in order to complete the tasks assigned to it. It can request help from a teacher, who then demonstrates a suitable action for the current state. Moreover, we would like our system to learn with just a small number of actions and requiring few demonstrations from the teacher. Therefore we propose the REX-D algorithm that extends REX [9] with the option to request demonstrations from a teacher.

1) The model: The REX-D algorithm builds upon a Markov decision process (MDP) that decides which action a to execute in the current state s to maximize the expected sum of rewards, using a learned transition model $P(s' \mid s, a)$.

The model is represented as a set of rules Γ that define the preconditions and effects of the actions. As we want to tackle

stochastic environments, we are using Noisy Indeterministic Deictic (NID) rules [11]. A NID rule r is defined as

$$a_r(\chi) : \phi_r(\chi) \rightarrow \begin{cases} p_{r,1} & : \ \Omega_{r,1}(\chi) \\ & \vdots \\ p_{r,n_r} & : \ \Omega_{r,n_r}(\chi) \\ p_{r,0} & : \ \Omega_{r,0} \end{cases}, \qquad (2)$$

where

- a_r is the action that the rule represents,
- $\phi_r(\chi)$ are the preconditions for the rule to be applicable,
- $\Omega_{r,i}$ are the effects defining the set of state predicates that are changed with probability $p_{r,i}$ when the rule is applied,
- $\Omega_{r,0}$ is the noisy effect that represents any other, unmodeled, rare and complex effects,
- χ is the set of variables of the rule.

A NID rule represents one action, while each action may be represented by several rules. All the rules defining one action have disjoint preconditions $\phi_{r_{a,i}} \wedge \phi_{r_{a,j}} = \emptyset \mid \forall i, j$. Therefore, each state-action pair (s, a) is covered by just one rule r.

2) REX-D: The system can take two different strategies. One is to *explore* the state space to improve the model and achieve better rewards in the long term; the other is to *exploit* by executing the actions that maximize the reward with the current learned model. REX-D (Algorithm 1) has many similarities with the E^3 version of REX [9]. It explores the state space until it reaches a known state. Once in a known state, it plans using a MDP with the known parts of the model, and if a plan is found it executes it (exploitation). However, unlike REX, when no plan is found in a known state, instead of using planned exploration, REX-D requests a demonstration from the teacher. This has two advantages:

- Addition of new actions as needed: Actions do not have to be defined at the outset. When no solution exists with the set of actions available, a teacher demonstration is requested and a new action can be taught.
- Improved learning time: The original REX algorithm uses the planned exploration step to select which parts of state space to explore. As the state space is usually very large, a lot of exploration may be needed until a solution is found. Nevertheless, the teacher demonstrates optimal actions which already lead the system to those parts of the state space that will incur high rewards.

Below the different parts of the algorithm are presented:

3) Counting function for exploration: Lang et al. [9] propose a solution for the exploration-exploitation dilemma in relational worlds, taking advantage of the relational representation to reduce the number of samples before considering a state as known. They use the following context-based density formula for state-action pairs:

$$k(s, a) = \sum_{r \in \Gamma} |E(r)| \, I(r = r_{s,a}), \qquad (3)$$

where $|E(r)|$ counts the number of experiences that cover the rule, and $I(\cdot)$ is a function which is 1 if the argument evaluates to true and 0 otherwise.

Algorithm 1 REX-D

Input: Reward function R, confidence threshold ζ

Set of experiences $E = \emptyset$
Rule set $\Gamma = \emptyset$
$t = 0$
Observe state s_t
loop
  Update Γ according to E
  if $\forall a \in A : k(s_t, a) \geq \zeta$ **then** ▷ *If state is known*
    Plan from s_t using Γ
    if found plan **then**
      $a_t =$ first action of the plan
    else
      Request demonstration
      $a_t =$ action demonstrated
    end if
  else
    $a_t = \text{argmin}_a \, k(s_t, a)$
  end if
  Execute a_t
  Observe new state s_{t+1}
  Add (s_t, a_t, s_{t+1}) to E
  $t = t + 1$
end loop

4) The learner: The approach of Pasula et al. [11] is used to obtain the rule sets. It is a greedy algorithm that generates NID rules to explain past experiences of action executions. It optimizes the trade-off between the rules' accuracy and complexity, obtaining models that are compact and tractable. Note that this learner requires the state to be fully observable.

5) The planner: The probabilistic planner PRADA [8] is used to generate action sequences to maximize the expected reward given a state and a NID rule set. It can obtain solutions for stochastic environments in a limited amount of time.

B. Monitoring

The user performs a sequence of actions while being monitored by the system. The system has been previously trained and is able at each moment to compute the optimal plan to reach the goal from the current world state. Each action executed by the user is validated as compatible with the current optimal plan or as diverging from optimality but still being recoverable. The latter means that the system is able to compute an alternative plan to reach the goal from the present world state, although this new plan has a larger number of steps than the optimal plan prior to execution of this action. Unrecoverable world states may also arise. Here, the planner is unable to find a plan to reach the goal, and the user is informed of this dead end. Algorithm 2 summarizes the monitoring mode of operation.

C. Teacher interactions

In both scenarios, learning and monitoring, the system has to recognize the actions executed by the user. During

Algorithm 2 Monitoring

Input: Goal oriented reward function R, model Γ

```
t = 0
Previous plan p_t = ∅
loop
    t = t + 1
    Observe state s_t
    Plan from s_t using Γ
    if plan p_t is found then
        Recommend first action of p_t
        if R(p_t) > R(p_{t-1}) then
            Signal good progress
        else
            Signal bad progress
        end if
    else
        Signal dead end
    end if
end loop
```

learning, when a demonstration is requested the action has to be recognized either as new or as already known. When monitoring, status signals are emitted after the user has executed an action. The manipulation recognition module continuously checks for new actions, and informs the decision maker when a new action gets executed.

V. The VR System

Virtual Reality plays the role of the world whose states are captured by perception and where state-modifying actions take place. To this end, the VR environment provides a dynamic representational space corresponding to the experimental settings (LABEX and AUTAS, see Section VI) which can be interactively modified. State descriptors, basically in terms of objects identifiers, poses and touching relations, are fed both to the SEC generation module and the DM. In turn, the DM provides action commands that are interpreted and executed in the VR environment (Figure 1). For further details on this system, please refer to [12].

The VR hardware consists of a multi-screen stereoscopic rear projection installation including three projectors and the corresponding rendering and managing servers. Interaction is achieved by using a wireless dataglove together with an ultrasonic tracking system for absolute positional tracking of the dataglove basis at the wrist. The dataglove, which is equipped with resistive bend-sensing technology to provide detailed readings of joint angles, keeps track of finger and wrist inclinations and sends these data to the VR system via a Bluetooth connection.

The basic structure of the underlying active object-oriented database is an extended scene graph (not to be confused with the equally-named but distinct concept introduced in Sec. III-A). The nodes in the graph represent single objects with their children being the constituents of the parent object, and with the root node representing the whole scene. The nodes contain the information to render the associated parts of the scene, simple spatial relations and geometries in its most basic form. Extensions are active properties added to a node in order to extend its capabilities.

The changing world state is updated in the database thanks to its ability of signaling changes made to its data. An advanced scripting language (SOML) supports the access to functions and properties of the core system and all components. SOML is capable of defining its own classes with interfaces for properties and functions. From the user's point of view, the most relevant function of SOML scripts is to execute the actions determined by the DM, taking temporarily control over the system and performing all the necessary state changes, taking care of the involved kinematics, and displaying the corresponding animations.

The user, equipped with polarized glasses for visual 3D immersion and wearing on the right hand the data glove described above, is able to interact with the scene, by touching and grasping the existing objects, moving them around, and releasing (or dropping) them. To this end, the VR system evaluates the data streams from the tracking system and the data glove and transforms them into the coordinate system of the scene, providing the 3D position and the orientation of the hand as well as finger postures, with respect to the same reference frame the virtual scene is referred to. Thus hand poses and collisions between the hand and other objects in the scene can be interpreted, enabling in this way to recognize grasping and releasing gestures. A virtual representation of the hand is displayed on screen, thus allowing for real time interactions with the virtual objects that are displayed and providing the user with a powerful clue for guiding their manipulative actions. With these tools, the user is able to grasp and release (or drop) objects in the scene in a very natural and intuitive way.

The system works either in teaching or in execution mode. In the first mode, a sequence of actions is performed by the user, who acts as a teacher that demonstrates actions to the system. During this phase, the user remains in control of the hand, which is symbolized by a natural human skin color of the virtual hand. In the second working mode, the system tries to plan with the rules learned so far and to execute the actions that are produced by the DM. Once the DM comes up with an action, the color of the virtual hand turns blue, thus signaling automatic execution. Control of the virtual hand is taken away from the user and the requested action is carried out automatically. If the DM reaches a dead end and does not know how to proceed (recall Section IV), control is returned to the human teacher, and the skin color of the virtual hand returns to a natural tone. At this moment, the user is able to perform the next action, and the DM refines its rule-base, as explained above.

VI. Experiments Performed on the VR Platform

A. LABEX: Monitoring of Human Performance

The LABEX scenario represents a laboratory environment in the International Space Station (ISS) where a scientific experiment takes place. The setting includes two cupboards, one of which should remain closed as long as possible in

order to keep the temperature constant (TCU), and the other one has compartments at different temperatures (STO), plus some experimental containers (EC) which can be transported on trays from and to the cupboards and the global deposit (see Figure 3).

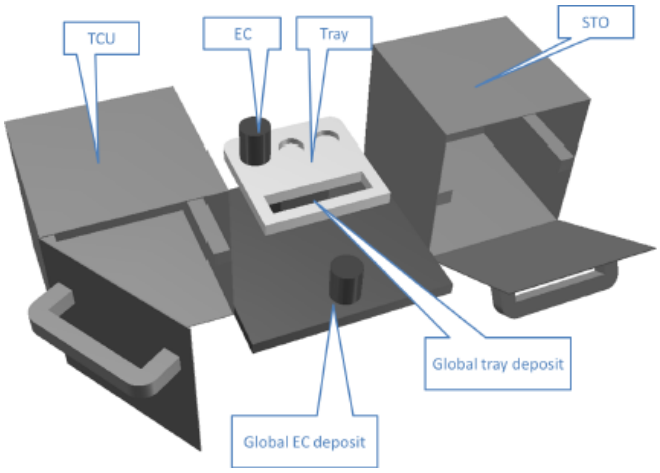

Fig. 3. LABEX scenario

Possible actions in this scenario include mainly opening and closing the doors of the cupboards, displacing the trays and extracting or inserting the ECs in the trays. The system is not intended for automatic execution of the actions, but for monitoring human performance along the experiments. The LABEX setting fits quite well to test the monitoring capabilities of the IntellAct system as the human operators in the ISS have to comply with well-defined sequences of actions in the laboratory with no room for interpretation and divergence.

The user, virtually situated in the Columbus module of the ISS, performs a set of manipulating actions. As described in Section V, the use of the dataglove enables detection of grasping and releasing gestures and thus interpretation of door opening and closing, tray transporting and EC handling actions in a natural way. As long as the user adheres to a trained sequence as the one above, a green traffic light appearing on one side of the scene signals correct performance. Non-optimal but still recoverable actions trigger a yellow traffic light. It is the task of the DM to produce a new plan that is still capable of reaching the goal from the current situation created by this action. If no such plan can found, i.e., if the goal cannot be reached anymore, for example by dropping an EC out of reach, the traffic light turns red.

B. AUTAS: Learning Assembly Sequences

The AUTAS scenario consists of the Cranfield assembly benchmark, whose parts are depicted in Figure 4.

The sequence in which the parts have to be assembled is conditioned by the precedence constraints in the assembly. These constraints have to be learned from sequences leading to successful completions of the assembly.

The user is virtually immersed in an industrial environment, and stands before a table with the parts of the

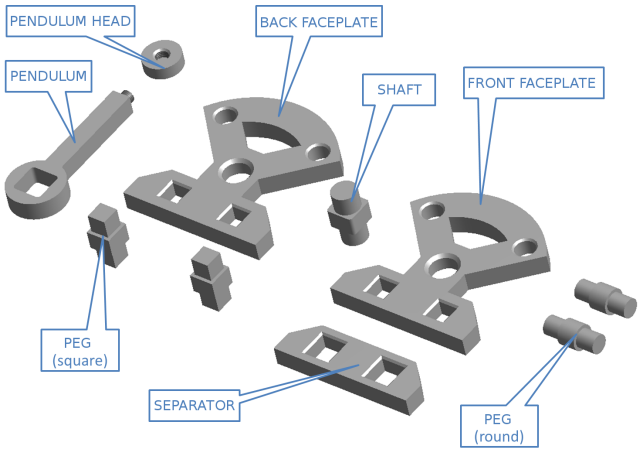

Fig. 4. Cranfield assembly parts

assembly scattered around. First the user demonstrates a whole assembly sequence by using the dataglove to grasp the individual parts and putting them into place. At this point, the rule learning module of the DM has learned a first set of rules which are usually far from being correctly defined, mainly as for their respective preconditions. The system then enters a rule-refinement/planning/execution loop (we consider the perception of the new states implicitly in this loop), where the DM makes a new plan with the available rules, executes the first action of this plan, and if it fails because the action is not yet applicable to the current state it requires intervention by the teacher. The user performs the action that should be executed at this point and the learning module updates the rule base by introducing additional preconditions to the affected rule. Figure 5 presents an example of the first two executions of this loop after the teacher's initial demonstration of the whole sequence.

First episode (teacher's demonstration of full sequence)	Second episode	Third episode
placepeg1(B1,FB_HB1)	placepeg1(B1,FB_HB1)	placepeg1(B1,FB_HB1)
placepeg2(B2,FB_HB2)	placefaceplatefront(FF)	placepeg4(B4,FB_HB4)
placepeg3(B3,FB_HB3)	placepeg4(B3,FB_HB3)	placeseparator(SP)
placepeg4(B4,FB_HB4)	placefaceplatefront(FF)	placepeg2(B2,FB_HB2)
placeshaft(SH,FB_HSH)	placeshaft(SH,FB_HSH)	placeshaft(SH,FB_HSH)
...	...	...

Fig. 5. Demonstration of the whole sequence of the teacher and the two first rule-refinement/planning/execution loops. Failed actions are displayed in red.

Furthermore, the learning module allows also to update the rule probability based on the observed action outcome. This enables the probabilistic planner to use this probability to select the next action.

C. Experimental results

We have conducted experiments in the two aforementioned modalities and settings. LABEX has been mainly used for automatic monitoring of human activity, which admits a qualitative evaluation (please refer to the videos). On the contrary, learning assembly sequences can also be evaluated quantitatively in terms of the number of requested teacher's

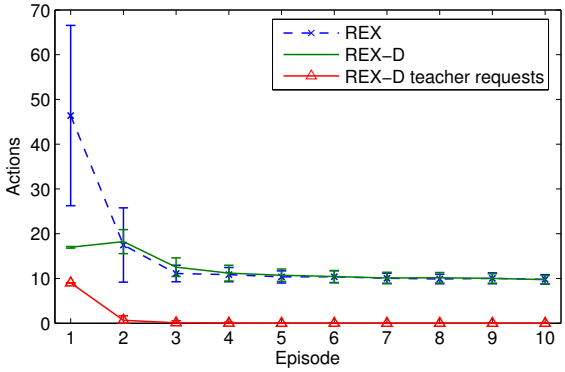

Fig. 6. Deterministic AUTAS scenario. REX-D starts with no prior knowledge, while REX starts with the list of actions available (but not their effects nor preconditions). The results show the mean and standard deviations obtained over 100 runs.

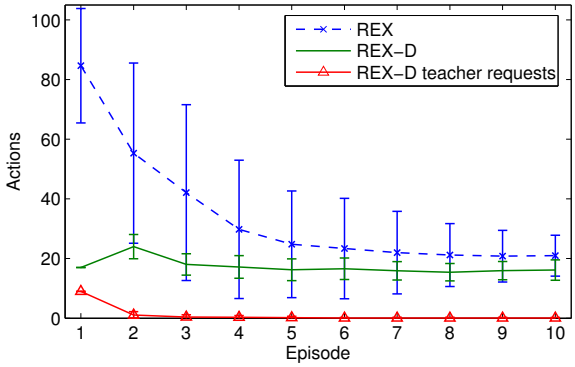

Fig. 7. Noisy AUTAS scenario. Actions have a success ratio of 60%. REX-D starts with no prior knowledge, while REX starts with the list of actions available (but not their effects nor preconditions). The decision maker was limited to executing 100 actions per episode. The results show the mean and standard deviations obtained over 100 runs.

demonstrations. Thus we concentrate here on presenting results for different experiments performed in the AUTAS scenario.

We emulate the functioning of a real robotic system by incorporating failures in the execution of an action. The source of such failures is either related to perception noise (i.e., during the segmentation and recognition processes, consisting in misidentified parts or in missing objects) or to action noise (during robot execution, e.g. failed grasps or insertions, dropped parts, or dead ends). In all cases, the result is that the outcome of an action is not as expected, and additional demonstrations by the teacher may be required to learn the task. Figure 6 shows the number of actions required to complete the Cranfield test in the deterministic case, whereas in Figure 7 the stochastic case is presented, with a success ratio for each action of 60%. REX-D clearly outperforms REX during the initial steps, where teacher demonstrations can save a lot of exploration actions. Moreover, the stochastic case also shows other improvements of having demonstration requests in complex problems. When the system doesn't know how to overcome unexpected results from an action, the teacher can provide an optimal demonstration that leads to the important part of the state space, while REX may have to explore suboptimal paths. Also note that the REX-D algorithm requires just a few teacher interactions after the first episode, which means that exploring is usually enough to complete the model, and only special cases require further help.

The the system can cope with execution contingencies that require new actions to be taught, and that enable probabilistic planning. For example, while typically a peg is expected to be found in a vertical position (as a result of a previous positioning operation on the table), it might incidentally be horizontal. In such cases, new actions can be taught such as "put_vertical" or "grasp_horizontal", and alternative action courses with different success probability may arise.

Teaching new actions also enables recovery from dead ends such as a premature placement of the separator before the corresponding pegs have been placed ("remove_separator"). Figure 8 illustrates the results obtained

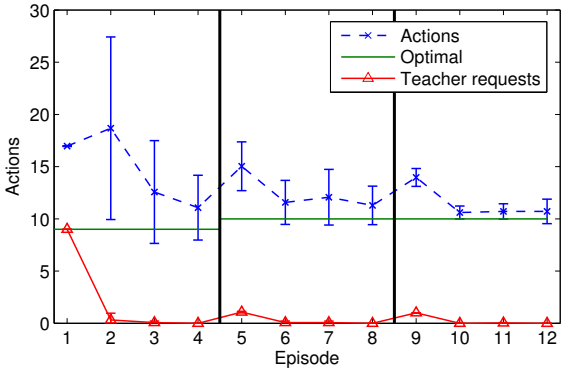

Fig. 8. AUTAS scenario. Episodes 1–4 show the standard AUTAS scenario. Episodes 5–8 exhibit a peg in an horizontal position, which has to be repositioned vertically before placing it. Episodes 9–12 start with the separator and no pegs placed, so the robot has to remove the separator in order to place the pegs, and then place the separator again. Therefore, during episodes 5–12 an additional action has to be performed. The learner starts with no prior knowledge. The results show the mean and standard deviations obtained over 50 runs.

for different variants of the task, requiring the teaching of the two aforementioned new actions (the first four episodes correspond to the standard Cranfield scenario, the next four to the case where a peg lies initially in a horizontal position and the new repositioning action has to be learned, and the last four to the case of separator placement without pegs). Of course, non-recoverable dead ends like unreachable objects or occlusions need to be recognized as such.

Having the teacher in the loop permits the further specialization of previously-taught actions. For example, depending on the available workspace, some types of grasps that are otherwise recommended for stability reasons are no longer possible: in Figure 9, the previous placement of the pendulum does not allow the separator to be placed while holding it at its center; it has to be held at one side. This gives rise to two specializations of "placeseparator()", as shown in Figure 10.

VII. Conclusions

A system that allows a robot to learn manipulation tasks has been presented. It includes a Decision Maker for the

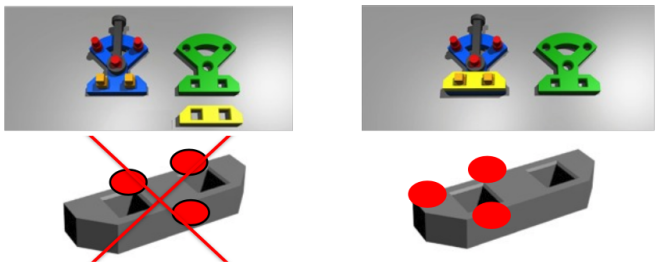

Fig. 9. While assembling the Cranfield benchmark, the pendulum has been placed before the separator. If the latter is grasped around the center as shown on the left, the fingers of the gripper would collide with the pendulum. Thus, it has to be grasped at one side, as shown on the right.

ACTION: placeSeparatorGraspCentral(X)
PRE: -pendulumPlaced(Y), ...
EFF 0.75: separatorPlaced(X), ...
EFF 0.25: noise

ACTION: placeSeparatorGraspSide(X)
PRE: pendulumPlaced(Y), ...
EFF 0.43: separatorPlaced(X), ...
EFF 0.57: noise

Fig. 10. The two grasp-conditioned specializations of the "placeseparator()" action. Observe the different preconditions, as well as the different success probabilities.

high-level learning and reasoning, a SEC module that obtains information about manipulations being executed, and a Manipulation Recognition module that recognizes the actions performed by the robot or a person. All these modules have been integrated in a way allowing a teacher to interact with the system to demonstrate how to execute various tasks. We have shown how the system can learn new tasks, execute the required manipulations to complete them, cope with unexpected contingencies, and monitor a person executing an already learned task. While the environment and the robotic effector are currently simulated in virtual reality, this work is targeted at real robotic platforms, which is work in progress.

The decision maker uses the novel REX-D algorithm, a relational reinforcement learning method that can request demonstrations from a teacher. This allows the decision maker to start with no previous knowledge about the actions available, as demonstrations of them are requested whenever they are needed. Moreover the combination of learning relational models and the help of having demonstration requests allows learning with just a small number of action executions.

We have shown that the system can also be used to monitor user actions once it has learned the task. Using the SECs and the manipulation recognition module, actions executed by the user are recognized, and the decision maker is able to give recommendations based on the current state, and to signal warnings whenever the actions are not leading to the goal.

Videos illustrating the system and showcasing key features can be found at `http://www.iri.upc.edu/groups/perception/ALOMS/`.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) under grant agreement no. 269959, IntellAct. D. Martínez is also supported by the Spanish Ministry of Education, Culture and Sport via a FPU doctoral grant (FPU12-04173).

References

[1] A. Agostini, C. Torras, and F. Wörgötter. Integrating Task Planning and Interactive Learning for Robots to Work in Human Environments. In *International Joint Conference on Artificial Intelligence*, pages 2386–2391, 2011.

[2] E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter. Learning the semantics of object-action relations by observation. *International Journal of Robotics Research*, 30(10):1229–1249, 2011.

[3] E. E. Aksoy, A. Abramov, F. Wörgötter, and B. Dellen. Categorizing object-action relations from semantic scene graphs. In *International Conference on Robotics and Automation (ICRA)*, pages 398–405, 2010.

[4] R. Brafman and M. Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2003.

[5] L. P. Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *International Conference on Robotics and Automation*, 2011.

[6] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, 2002.

[7] J. Kober and J. Peters. Reinforcement learning in robotics: a survey. *International Journal of Robotics Research*, pages 579–610, 2012.

[8] T. Lang and M. Toussaint. Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research*, 39:1–49, 2010.

[9] T. Lang, M. Toussaint, and K. Kersting. Exploration in relational domains for model-based reinforcement learning. *Journal of Machine Learning Research*, 13:3691–3734, 2012.

[10] L. Li, M. Littman, T. Walsh, and A. Strehl. Knows what it knows: a framework for self-aware learning. *Machine learning*, 82(3):399–443, 2011.

[11] H. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29, 2007.

[12] J. Rossmann, C. Schlette, and N. Wantia. Virtual Reality in the Loop – Providing an Interface for an Intelligent Rule Learning and Planning System. In *Semantics, Identification and Control of Robot-Human-Environment Interaction (Workshop at the International Conference on Robotics and Automation), 2013*, pages 60–65, 2013.

[13] J. Sung, B. Selman, and A. Saxena. Learning sequences of controllers for complex manipulation tasks. In *International Conference on Machine Learning*, 2013.

[14] T. Walsh. *Efficient learning of relational models for sequential decision making*. PhD thesis, Rutgers, The State University of New Jersey, 2010.

[15] T. Walsh, I. Szita, C. Diuk, and M. Littman. Exploring compact reinforcement-learning representations with linear regression. In *Conference on Uncertainty in Artificial Intelligence*, pages 591–598, 2009.

[16] T. J. Walsh, K. Subramanian, M. L. Littman, and C. Diuk. Generalizing apprenticeship learning across hypothesis classes. In *International Conference on Machine Learning*, pages 1119–1126, 2010.

[17] T.J. Walsh and M.L. Littman. Efficient learning of action schemas and web-service descriptions. In *AAAI Conference on Artificial Intelligence*, pages 714–719, 2008.

[18] Q. Yang, K. Wu, and Y. Jiang. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171, 2007.

[19] H. H. Zhuo, T. Nguyen, and S. Kambhampati. Refining incomplete planning domain models through plan traces. In *International Joint Conference on Artificial Intelligence*, 2013.