Efficient Interactive Decision-making Framework for Robotic Applications

Alejandro Agostini[a,*], Carme Torras[b], Florentin Wörgötter[a]

[a]*Bernstein Center for Computational Neuroscience, 37073 Göttingen, Germany*
[b]*Institut de Robòtica i Informàtica Industrial (CSIC-UPC), 08028 Barcelona, Spain*

Abstract

The inclusion of robots in our society is imminent, such as service robots. Robots are now capable of reliably manipulating objects in our daily lives but only when combined with artificial intelligence (AI) techniques for planning and decision-making, which allow a machine to determine how a task can be completed successfully. To perform decision making, AI planning methods use a set of planning operators to code the state changes in the environment produced by a robotic action. Given a specific goal, the planner then searches for the best sequence of planning operators, i.e., the best plan that leads through the state space to satisfy the goal. In principle, planning operators can be hand-coded, but this is impractical for applications that involve many possible state transitions. An alternative is to learn them automatically from experience, which is most efficient when there is a human teacher. In this study, we propose a simple and efficient decision-making framework for this purpose. The robot executes its plan in a step-wise manner and any planning impasse produced by missing operators is resolved online by asking a human teacher for the next action to execute. Based on the observed state transitions, this approach rapidly generates the missing operators by evaluating the relevance of several cause-effect alternatives in parallel using a probability estimate, which compensates for the high uncertainty that is inherent when learning from a small number of samples. We evaluated the validity of our approach in simulated and real environments, where it was benchmarked against previous methods. Humans learn in the same incremental manner, so we consider that our approach may be a better alternative to existing learning paradigms, which require offline learning, a significant amount of previous knowledge, or a large number of samples.

Keywords: Decision making, human-like task, logic-based planning, online learning, robotics

1. Introduction

Complex, partially autonomous machines or software systems have been part of our society for many years. They often coexist with us unnoticed, e.g., to control traffic, influence our shopping decisions, or to help us in our houses or working environments. By contrast, until recently, robots have been confined to perfectly controlled industrial environments because of their limited capabilities to adapt to variable conditions. This was due partly to the unbalanced progress of computer science and mechatronics. However, recent advances in sensing (e.g., computer vision [23]), acting (e.g., robotic manipulation techniques [15]), and reasoning (e.g., artificial intelligence (AI) [20]) have now allowed them to enter everyday life. For example, robots are capable of recognizing and manipulating a wide range of objects found in a human kitchen such as cups, doors, tables, knifes, or vegetables [3]. Hence, versatile service (kitchen) robots are about to become available. This might be particularly helpful for people with restricted mobility who need assistance to perform simple tasks such as setting a table. Other application domains may include hazardous environments or disaster areas, where human lives would be at serious risk and thus reliable robots could provide solutions.

However, a serious bottleneck still makes it difficult to develop robotic applications for these domains. In addition to the existing hard problems in perception analysis and action control, the lack of true autonomy prevents machines

*Corresponding author.
Email addresses: `aagosti@gwdg.de` (Alejandro Agostini), `torras@iri.upc.edu` (Carme Torras), `worgott@gwdg.de` (Florentin Wörgötter)

from behaving in a similar manner to humans. How can a robot "understand" the consequences of an action to achieve autonomy about whether to perform that action in a specific situation? At present, very few approaches are available to facilitate the teaching of robot actions and thus machines have difficulty in learning the cause-effect relationships for a specific action. This is puzzling because this type of teaching, either declarative based on explanations or implicit based on demonstrations, is arguably the most efficient manner in which humans learn the consequences of their actions. Thus, embedding robots in human environments requires the development of tools that can allow people who are not experts in robotics, such as home users, to instruct a machine to perform different human-like tasks. The goal of the current study is to provide a suitable tool for this purpose.

It is vital that human-robot interactions occur in a natural and efficient manner. This can be achieved by using a human-like declarative knowledge representation to specify the goal and any instruction for the ongoing task that a human may need to provide. Many robust and well-known AI techniques for decision making use a declarative knowledge representation, where one of the most widely used is logic-based planning [16]. This type of planning uses a logic-based knowledge representation in terms of predicates and propositional logic, which allows the description of the states and actions of a task using a representation that is very similar to natural human language. For instance, to indicate that a cup is on a table, we can simply define the predicate $ontable(cup)$, which takes the value *true* if there is a cup on a table, and *false* otherwise. Similarly, the action of picking a cup from a table is coded as $pick(cup, table)$. In this manner, any human instruction can be easily transformed into a logic notation that specifies the human-robot interaction, thereby making the definition of the planning problem simple and straightforward.

A logic-based planner uses a set of planning operators (POs) that code cause-effect relationships in terms of propositions and logical predicates. These cause-effect descriptions comprise the changes that occur in a state with an action, and the necessary conditions that should be present in a state to permit these changes to occur. If the task is well known and it involves only a few state transitions, the usual strategy is to hand-code these operators. However, for robots that perform human-like tasks, hand-coding all the necessary POs can be a burden. Even for apparently simple tasks, many of the relevant aspects of the cause-effect descriptions that need to be encoded can be easily overlooked. Our proposed alternative is to learn these operators automatically from example state transitions. There are two approaches for learning operators. The first involves collecting samples in advance and learning offline. This approach may lead to some complications, because if the collected samples are not sufficient to learn all of the necessary operators, the robot may be forced to abandon the task completely during runtime due to missing or incomplete operators. The second approach learns operators online while the task is being executed under the guidance of a human, which is similar to the process employed when we teach our children. If a planning impasse is reached due to missing operators, the robot simply asks the human for the next action to execute and tries to continue from that point. This strategy prevents long task interruptions and facilitates adaptation to non-stationary changes in the environment.

In this study, we propose an interactive decision-making framework (iDMF), which integrates and adapts AI techniques so they can be used for planning, online learning of POs, and execution of human-like tasks. Our approach facilitates the learning of useful POs online from limited experience without task interruptions and without the need for any previous codification of POs. To make this possible, the iDMF interleaves planning and learning to improve the capability of the planner progressively as more operators are learned, thereby allowing the robot to make immediate use of any improvement in the set of POs. If the planner generates a plan, it is executed and monitored by the system after each action is executed. If the planner is not able to find a plan due to missing operators, the iDMF uses a human teacher to instruct the action that needs to be executed, which allows the continuous execution of the task.

To ensure that this approach works in an efficient manner, the robot needs to avoid spending substantial amounts of time during each planning-learning interaction and it should rapidly find the relevant conditions that need to be considered for each PO. This problem is addressed using a learning strategy that updates and evaluates a set of alternative operators *in parallel*, thereby making more efficient use of each learning experience. This approach tests many combinations of conditions that are potentially relevant to the successful execution of the operators. Among all of these alternatives, those that are most likely to lead to the successful execution of the operators are used for planning. To rapidly evaluate the relevance of the conditions, we developed a new probabilistic estimate, which compensates for lack of experience by producing confident estimates based on only a few examples.

In this study, we consider two applications. First, we use the game of Sokoban (box-pushing) to provide an in-depth understanding of all the algorithmic processing steps, where this game is directly analogous to many human-like tasks that require the movement of objects in an ordered manner. The robotic implementation of this game is simple and straightforward but less relevant compared with our second example, i.e., cup sorting and stacking by a humanoid robot. This example is more difficult to analyze but it is truly relevant to the domain of robotic applications, i.e., picking up objects and positioning them in unconstrained environments.

The remainder of this article is organized as follows. The next section summarizes related research. In Section 3, we provide a brief explanation of the iDMF in which the learning method is embedded, and descriptions of the basic notations used in this study. Section 4 explains our proposed method for the online learning of POs. In Section 5, we present evaluations of the capacity of the learning approach for improving the decision-making performance of the whole system. Finally, we discuss this study and give our conclusions.

2. Related Work

Recent AI techniques for learning and reasoning facilitate the description of the world in a manner that is shareable among humans [2, 5]. For instance, some learning from demonstration techniques [2] exploit the intuitive capacity of humans to demonstrate actions to teach robot POs in the form of pre- and post-conditions [27, 28, 4, 26, 18, 21]. Some of these techniques interleave planning and learning in human-like task applications [27, 28, 4, 22, 12, 26]. For example, in the methods described in [27] and [28], a set of POs is generated initially using examples, which are collected offline from the expert's solution traces of the task that needs to be executed. Next, the system refines these operators using traces collected from a simulated environment where a modified PRODIGY planner [25] is used for decision making and an adaptation of the version space method called OBSERVER is used to refine the POs. If the planner fails to find a plan because the set of operators is incomplete, it returns a failure signal and interrupts the execution of the task. In [4], a strategy for continuous planning and learning was proposed called TRIAL, which generates plans, executes them, and learns operators until the system reaches a planning impasse, when an external teacher is called to take control and complete the task. In [12], a system was proposed called EXPO, which uses PRODIGY as a baseline planner to improve knowledge in several domains where the initial knowledge may be up to 50 % incomplete. POs are learned using the results obtained from simulated environments and they are then available immediately for planning. Our approach is different because we strongly emphasize online learning of the required iDMF. The teacher intervention occurs during task execution by *only* providing the action that needs to be executed and returning the control of the task to the system after giving the instruction. This avoids the burden of providing representative sequences for learning in situations that the teacher needs to design.

The strategy that we employ for action instruction is similar to other types of teleoperation demonstration techniques [2]. The robot is provided with a set of capabilities (actions) in advance and the teacher simply indicates which of them to execute in a given situation. This can be achieved via a spoken command, such as in [21], or simply by letting the teacher select the action from a list presented by a graphical interface. After the action has been instructed, we allow the robot to execute the action and generate the state transition used for learning. By allowing the robot to execute the action, we simplify the *correspondence issues* [2] when identifying the mapping between the teacher and the learner, which facilitates the transfer of information. In particular, the strategy employed for teacher interaction is similar to that proposed in [18]. However, in contrast to that approach, the teacher does not perform any supervision of actions and does not provide attentional cues about relevant attributes of the world that should be present to execute a task successfully. This may increase the effort required to learn the task, but it also simplifies the effort of the teacher considerably. It should be noted that indicating all the relevant attributes of the world in every possible situation faced by the robot may be complicated, depending on the complexity of the application. To ease this burden, we propose a method for automatically learning the relevant attributes of the world, which should be present to allow the successful execution of a PO.

The proposed method is an extension of [1]. The most significant advances compared with [1] are an improved method for the generation and refinement of POs, and a thorough evaluation of the iDMF by comparing it with a widely known approach for the online learning of POs, i.e., the OBSERVER method [27]. In [1], we presented a detailed description of the PO evaluation strategy (i.e., the *density*-estimate, see Section 4.1), which is used in a simple strategy to generate the POs. In the current study, in Section 4.2, we present a new improved version of the learning mechanisms for generating POs. Furthermore, in [1], we only evaluated the performance of the system in real robot scenarios using simple tasks. In the present study, we include a more thorough evaluation of the system where we used simulated scenarios, which allowed us to test the system with different task complexities and to thoroughly compare its performance with that of the OBSERVER method. We also include new experiments performed in real robot scenarios.

3. iDMF

A general schema of the implemented iDMF is shown in Fig. 1. The main components of the framework are the *planner*, which is responsible for finding the sequence of actions require to achieve the goal of the task, and the *learner*, which generates and refines the POs. The planner and the learner use a declarative knowledge representation for reasoning and learning. For example, the *state* specified in the figure describes the environmental and the robot situations, which may include descriptors indicating that a cup is on the table, that a door is closed, or that the hand of the robot is empty. All of these state descriptors are derived from the raw signals obtained from the sensors of the robot, e.g., force signals or an image obtained from a camera using computer vision techniques for segmentation and object recognition. This is actually the role of the *perception* module.

The *actions* generated by the planner are instructions to the robot, which are specified in a declarative manner, where they may request the robot to grasp a cup from a table, open a door, cut some vegetables, or switch off a light. Similar to the symbolic state description, these symbolic actions should also be grounded to allow their actual execution. This is achieved by the *execution* module, which transforms the declarative instructions into actual motor commands for the robot actuators. Note that the perception and execution modules cluster all of the mechanisms for symbol grounding that are necessary for integrating the AI techniques into a robotic platform.

After providing a declarative description of the currently existing world state, the planner tries to find the sequence of actions that would transform the current state into a state that satisfies the goal specification, which is also provided in the same declarative manner. Thus, the planner uses the POs that specify the state changes produced when an action is executed. Using this information, the planner executes a reasoning process, which usually involves searching for the shortest sequence of changes that transforms the current state into a goal state. An example of a PO can be found in Section 3.1. Coding the changes that may occur in all possible states with all possible actions is not a simple task for applications with a large number of state transitions. Therefore, in our iDMF, the POs are learned automatically from experienced state transitions (*instance* in Fig. 1) using the learner. The learner, which is interleaved with the planner, generates and refines POs after every action is executed.

The planner tries to find a plan using the operators learned up to a given moment. If a plan is found, only the first action of the plan is sent to the execution module, instead of the whole plan. This allows an evaluation of whether the expected changes coded in the corresponding PO have actually occurred. If the expected effects have not occurred, the learner refines the executed PO by searching for the missing conditions that prevented the expected changes from occurring. After executing an action, the next state is generated by the perception module and used as a new initial state for planning. This *replanning* process permits the immediate use of the new knowledge generated by the learner and it allows an incomplete set of POs.

Another possible outcome for the planner is that no plan is found due to some missing operators. In this case, the planner sends a request to a human *teacher* who needs to specify the next action for the ongoing task. We prefer to use a human teacher because their inclusion in the planning-learning loop is simple, highly efficient, and straightforward for human-like tasks.

Finally, if the goal is reached, the planner generates the end-of-plan signal, and thus the task stops.

3.1. Basic Notations

In this section, we introduce the basic notations used in this study. We assume that there is a finite (or countably infinite) set of states $\mathscr{S}$ and a finite (or countably infinite) set of actions $\mathscr{A}$. A *state* s is described using a set of state descriptors, $\{d_i\}$, each of which comprises a logic predicate that describes the properties of objects or the relations between them. Predicates are logic formulas that map the object space onto *true* or *false* values. For instance, a predicate describing the fact that a cup is on the table could be $ontable(cup)$ and one indicating that the hand of the robot is empty could simply be $empty(hand)$. Note that this notation is very similar to natural human language, thereby making the interpretation of a particular state highly intuitive for a human.

An *action* a is described declaratively and it may take arguments representing the objects to which the action is applied and other parameters required for the action. For example, to instruct the robot to open a door, the action

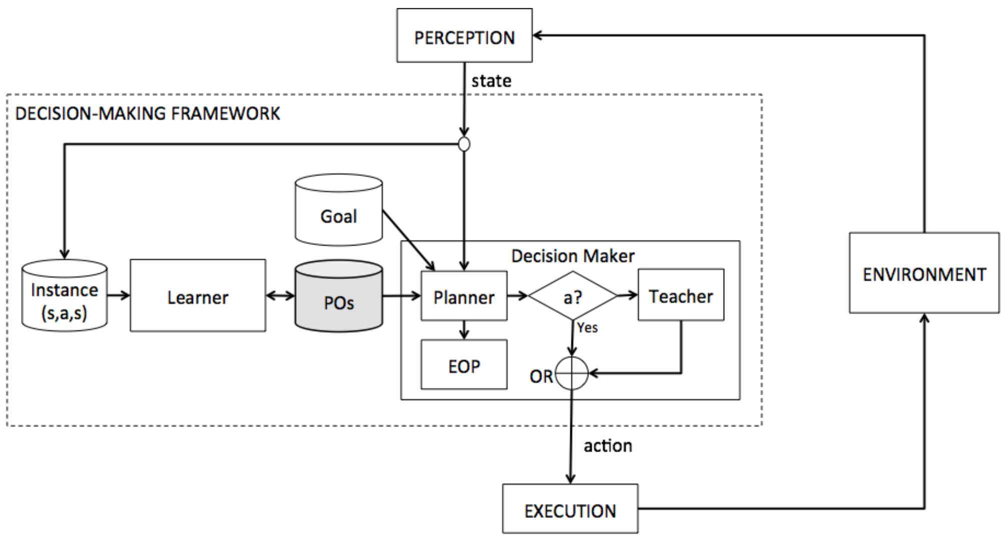


Figure 1: Schema of the iDMF.

could be coded as $open(door)$. Other possible actions could be $grasp(cup)$ to grasp a cup, $cut(potate, knife)$ to cut a potato with a knife, or $move(box, room1, room2)$ to move a box from room 1 to room 2.

In logic-based planning, the planner receives the description of the *initial state*, s_{ini}, and a *goal* description, g, which comprises a set of grounded predicates that should be observed after plan execution [11]. Using these elements, the planner searches for sequences of actions that would permit changes in s_{ini} that are necessary to reach the goal g. This search is performed using a set of POs, $\mathscr{R}$, each of which codes the expected changes after executing an action. A PO is represented as

$$r = \{p, a, e, P\}, \tag{1}$$

where p is the precondition [1], a is the action, e is the effect, and P is a probability estimate, which indicates how likely the effect e will be obtained every time that p is observed in a state and action a is executed,

$$P = \Pr(e|p, a). \tag{2}$$

A PO encodes the changes that should be observed when an action is executed. The precondition p contains the initial values of the state descriptors that are changed by the action, while the effect e contains their final values. The action a describes the action that needs to be executed. In addition to the changed descriptors, the precondition should also contain the state descriptors that do not change with the action, but which are necessary to allow the changes to occur. For example, a simple PO for an action of opening a door may have the following parts:

$$\begin{aligned} p &= \{not(opened(door)), not(islocked(door))\}, \\ a &= open(door), \\ e &= \{opened(door)\}, \end{aligned} \tag{3}$$

where $not(opened(door))$ is a predicate that takes a value *true* when the door is not opened, $not(islocked(door))$ is a predicate that takes a value *true* when the door is unlocked, and $opened(door)$ is a predicate with a value *true* when

[1] For clarity, we represent the precondition as a set of predicates that should be true to apply the PO. This specific representation can easily be transformed into the representation needed for any selected planner, e.g., a conjunctive normal form where the literals are these predicates. For the precondition and state representations, we make the closed-world assumption, i.e., all predicates that are not explicitly considered are assumed to be false.

the door is opened. In this case, the state descriptor that changes with the action is $not(opened(door))$. However, for this change to occur, it is also necessary that the door is unlocked, which is a fact specified by the descriptor $not(islocked(door))$ that remains unchanged by the action.

A PO provides a compact description of the consequences of the actions. Thus, among all the descriptors that are relevant to complete a task, i.e., those that should be considered in the state description, each PO only considers those descriptors that are relevant for predicting the effect of a particular action. For instance, if the PO in (3) is used in a task that also requires other actions, e.g., leaving a box on a table, a descriptor such as $ontable(box)$ will also be needed in the state representation but neglected in the PO of opening a door. This is because the descriptor $ontable(box)$ is not relevant to opening the door. In general, the number of descriptors used in the state representation is much larger than the number of descriptors used in a particular PO. This is a fundamental difference compared with the traditional representation of the consequences of actions as state transitions used by most planning methods based on dynamic programming [16]. In this sense, a PO performs a generalization over the space of state transitions where the consequences of an action (i.e., changes) are the same. This may result in a drastic reduction in the search space size compared with traditional dynamic programming methods.

In the proposed framework, not all of the POs are available for planning and some will only be handled by the learning method. We make this distinction when needed.

The training instances for learning comprise state transitions of the form:

$$I_t = (s_t, a_t, s_{t+1}), \tag{4}$$

where t is the time step, s_t is the state before executing the action, a_t is the executed action, and s_{t+1} is the state after executing the action. A *positive* instance $I^+_{t,i}$ for the PO $r_i = \{p_i, a_i, e_i, P_i\}$ is defined as that where all of the predicates in the precondition are observed in the state before the execution of the action, the action coincides with the action of the state transition, and the predicates in the effect are observed in the state after the execution of the action:

$$\{I^+_{t,i} | p_i \subseteq s_t, a_i = a_t, e_i \subseteq s_{t+1}\}. \tag{5}$$

A positive instance represents a state transition obtained from a successful execution of the PO, i.e., an execution where the expected changes were obtained. By contrast, a *negative* instance $I^-_{t,i}$ represents a state transition where the PO was applied but the expected changes did not occur, i.e., the precondition was fulfilled in the state before the action execution and the action was the same as the state transition, but the effect of the PO was not observed in the state after executing the action:

$$\{I^-_{t,i} | p_i \subseteq s_t, a_i = a_t, e_i \nsubseteq s_{t+1}\}. \tag{6}$$

We say that an operator r_i *covers* an instance when $p_i \subseteq s_t$ and $a_i = a_t$.

3.2. iDMF Algorithm

The algorithmic description of the iDMF is presented in Algorithm 1. Note that the initial set of POs could be the empty set or it may contain previously learned/defined operators.

4. Learning Planning Operators

The main difficulty when learning the POs is finding relevant descriptors that permit the coded changes to occur, but that are not affected by executing the action. To address this problem, we need to define a method to calculate the conditional probability (2) and a method for generating new POs by considering different combinations of potentially relevant descriptors in their preconditions. The following sections focus on these two aspects.

Algorithm 1 iDMF

```
Specify goal $g$
Initialize PO set $\mathscr{R}$ (e.g., $\mathscr{R} \leftarrow \varnothing$)
$s_{t-1} \leftarrow \varnothing$
$a_{t-1} \leftarrow \varnothing$
**repeat**
  Get current state $s_t$ {Perception module}
  Generate training instance $I_{t-1} = (s_{t-1}, a_{t-1}, s_t)$
  $\mathscr{R} \leftarrow$ LEARNER($\mathscr{R}, I_{t-1}$) {update PO set with current instance}
  $s_{ini} \leftarrow s_t$
  $A =$ PLANNER $(s_{ini}, \mathscr{R}, g)$ {find a plan $A$ to reach goal $g$ from current state $s_{ini}$}
  **if** $g \subseteq s_{ini}$ **then**
    End of plan
  **else**
    **if** $A = \varnothing$ **then**
      $a_t \leftarrow$ TEACHER $(s_{ini}, g)$ {request an action from the teacher for the ongoing task}
    **else**
      $a_t = A(1)$ {first action of the plan}
    **end if**
    Execute $a_t$ {Execution module}
    $t \leftarrow t+1$
  **end if**
**until** End of plan
```

4.1. PO Evaluation

One of the main problems when evaluating a PO using the probability in Eq. (2) is the high uncertainty of the estimates when they are calculated from a small pool of experience samples. This implies that if the lack of experience is not considered, the approach may incorrectly produce large premature estimates, which will degrade the performance of the system, mainly in the early stages of the learning.

To prevent biased premature estimates, we use the *m*-estimate formula [7, 10] to estimate the probability (2) as follows:

$$P = \frac{n^+ + mc}{n^+ + n^- + m}, \tag{7}$$

where n^+ is the number of positive experience instances covered by the PO, n^- is the number of negative instances covered, c is the *a priori* probability of a positive instance, and m is a parameter that regulates the influence of c. The parameter m plays a role as the total number of nonexperienced instances covered by the PO. For a given c, a larger value of m indicates the lower influence of the experienced instances on the probability and the estimate is closer to c. This allows the influence of the initial experiences to be regulated with m, thereby preventing large premature estimates. To illustrate how this regulation occurs, we use the extreme case of setting $m = 0$, i.e., no nonexperienced instances are considered, which leads to the traditional frequency probability calculation:

$$P = \frac{n^+}{n^+ + n^-}, \tag{8}$$

where an estimate made using only a couple of positive instances may have a 100 % chance of being positive, regardless of the uncertainty of the instances that still need to be experienced. By contrast, if we define a larger value of m, the influence of the observed instances decays and the estimate is closer to c. The setting of m is defined by the user according to the specific classification problem. One known instantiation of the m-estimate is to set $m = 2$ and $c = 1/2$, where we have the Laplace estimate [7]:

$$P = \frac{n^+ + 1}{n^+ + n^- + 2}, \tag{9}$$

which is used widely in well-known classification methods such as CN2 [8]. One drawback of the original *m*-estimate is that it does not provide a way of regulating the influence of *m* with more experience since the value of *m* is constant. This degrades the accuracy of the estimate as learning proceeds, where it is worse for larger values of *m*, which makes the estimate biased toward *c*. To avoid this problem, we propose using a variable *m*, which comprises an estimate of the number of instances $\hat{n}^{\emptyset}$ that still need to be experienced before we can consider the estimate as confident,

$$P = \frac{n^{+} + \hat{n}^{\emptyset} c}{n^{+} + n^{-} + \hat{n}^{\emptyset}}. \tag{10}$$

Equation (10) can be interpreted as the conventional frequency probability calculation (8), where each inexperienced instance contributes a fraction *c* of a positive sample. If the task under consideration allows us to know the total number of instances covered by the PO, n^T, we can calculate the exact number of nonexperienced instances and use it as the number of experiences that are needed before we can consider the estimate as fully confident:

$$\hat{n}^{\emptyset} = n^T - n^{+} - n^{-}. \tag{11}$$

In this case, by using (11) in (10) and reformulating, we obtain,

$$P = c\left(1 + \frac{(1-c)}{c}\frac{n^{+}}{n^T} - \frac{n^{-}}{n^T}\right), \tag{12}$$

which we call the *density*-estimate. It should be noted that with this equation, the probability *P* changes as a function of the density of the positive and negative instances, rather than as a function of the relative frequencies. Low densities of instances will produce low variations in the probability, thereby preventing premature large estimates when few examples are collected. As learning proceeds, the influence of the densities will become larger and the probability estimate will tend to the actual probability. For example, when all of the instances have already been experienced, we have $n^T = n^{+} + n^{-}$ and Eq. (12) is equal to (8).

Note that the problem of evaluating a PO can be treated as a classification problem where instances are classified as positive or negative. In this sense, Eq. (12) is the probability estimate for the positive class, P^{+}, whereas the estimate for the negative class is

$$P^{-} = c\left(1 + \frac{(1-c)}{c}\frac{n^{-}}{n^T} - \frac{n^{+}}{n^T}\right). \tag{13}$$

We can easily generalize the *density*-estimate for the multi-class case as

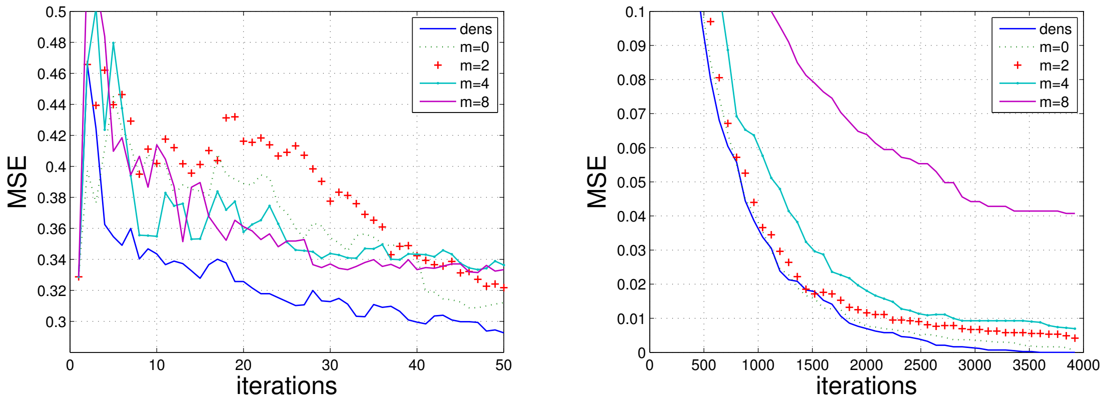
$$P^{i} = c\left(1 + \frac{(1-c)}{c}\frac{n^{i}}{n^T} - \sum_{\substack{j=1 \\ j\neq i}}^{K}\frac{n^{j}}{n^T}\right), \tag{14}$$

where *i* is the class that needs to be evaluated and *K* is the number of classes.

In the context of planning, the probability in Eq. (13) represents the probability of obtaining any effect other than that coded in the PO, and thus when defined together with the probability in Eq. (12), we obtain a binary probability distribution over the expected effects. In the general case of Eq. (14), the *density*-estimate can be used to calculate the probability distribution over different effects for the same PO, and not exclusively in a binary manner. However, given the scope of this study, we evaluate a PO using the binary approach.

4.1.1. Performance of the Density-Estimate

In order to evaluate the performance of our *density*-estimate, we use the binary classification problem called Monk's problem number 2 [24]. In this problem, the input space has six attributes, each of which represents a feature of an artificial robot: head shape, body shape, is smiling, holding, jacket color, and has tie. The output is 1 only when two attributes have their first values, and 0 otherwise. We consider this problem because it is a complex classification problem that poses difficulties for many known classification methods and it is directly analogous to the binary approach for evaluating the outcome of a PO.



(a) Early stages of learning (b) Long term

Figure 2: Comparison of the performance of our *density*-estimate and that of the *m*-estimate.

For the evaluation, we define a classification rule H_i as a set of attribute-values from Monk's problem number 2. Each rule covers a number of positive instances, n^+, and a number of negative instances, n^-. The total number of instances covered by a classification rule is denoted by n^T. To obtain the class of a given instance, from all the classification rules covering the instance, we select the rules with the highest P^+ and P^-, which are calculated using Eqs. (12) and (13), respectively. The classification for that instance is thus the class with the highest probability. Two new classification rules are generated each time a misclassification occurs by refining the failed rule with one more attribute-value, which is selected randomly from the attribute-value set.

We compare the results obtained using the original *m*-estimate, with $m = 0, 2, 4, 8$, and our *density*-estimate to calculate the probability of a class. Note that for $m = 0, 2$, we obtain (8) and (9), respectively. To ensure a fair comparison, we set $c = 1/2$ in all cases. Training instances are selected randomly in the input space with uniform probability. After each training iteration, a test episode is run, which involves calculating the classification error for every input in the input space. The results represent the average of the classification errors from 10 runs for each case considered.

The results show that when few instances are experienced (Fig. 2(a)), the performance of the conventional *m*-estimate appears to improve as the value of *m* increases. However, this result is reversed as learning progresses (Fig. 2(b)) due to the inability of the original *m*-estimate to compensate for the component introduced by the large *m*. By contrast, our proposed method compensates for the effect of the lack of experience when estimating the probability by producing more confident estimates to outperform the original *m*-estimate at all stages of the learning process.

4.2. PO Generation

In the previous section, we described how to evaluate a PO using the *density*-estimate. In this section, we show how POs can be generated using descriptors with the highest probabilities of the observed changes.

It is important to generate POs with high probabilities as well as POs that can be applied to a large number of states. The generalization of a PO depends mainly on the number of descriptors considered in its precondition *p*. If there are fewer state descriptors in *p*, the PO can be applied in a larger number of states. Thus, a PO will facilitate the prediction of the result when executing the PO action in all of these states, even when many of them have not been yet experienced. This is particularly important in high-dimensional problems where the number of possible states is very large.

High-dimensional problems also present serious difficulties when learning the POs. In this problem, testing all of the possible combinations of state descriptors is intractable and a special strategy should be employed that explores only the potentially relevant combinations. We assume no previous knowledge about the relevant regions of the state space, so the system should allow the generation of any possible combination of descriptors while still limiting the search.

To reduce the time invested in learning, the system only generates POs when a plan cannot be generated due to missing POs or when the execution of a PO has an unexpected effect.

4.2.1. Generation from an Action Instruction

Missing POs may prevent the planner from generating a plan, thereby triggering an action instruction request to the teacher. After executing the instructed action, a_{inst}, a new PO is generated to fill the knowledge gap that generated the instruction request. First, the changed descriptors are extracted from the observed state transition,

$$p_{change} = \{d_i \in s_t | d_i \notin s_{t+1}\},$$
$$e_{change} = \{d_j \in s_{t+1} | d_j \notin s_t\},$$

where p_{change} and e_{change} are the sets of the initial and final values of the changed descriptors, respectively, s_t is the state before and s_{t+1} is the state after executing the action. Then, a new PO, r_{inst}, is created by using p_{change}, a_{inst}, and e_{change} as the precondition, action, and effect parts, respectively. The new PO becomes available immediately for planning.

However, the new PO only considers the changed descriptors in its precondition, so some unchanged causative descriptors may be missing. In this case, the execution of the newly generated PO may have an unexpected effect.

4.2.2. Generation from Unexpected Effects

An unexpected effect triggers the refinement of the precondition of the executed PO, r_{exe}, to search for those descriptors that are necessary for the changes coded in the PO but that are still missing from the precondition. To find these descriptors quickly, our strategy memorizes the most accurate refinements for each of the POs that are available for planning and refines them in parallel using a general to specific approach, which searches for relevant combinations of descriptors. Next, the most accurate PO obtained from these refinements replaces the failing PO. Our learning approach can be viewed as expanding multiple search trees that are independent of each other. This allows the parallel testing of combinations of descriptors that belong to different regions of the state space (possibly disjoint), thereby increasing the chances of finding those that are relevant. The refinement process starts by bringing together all the POs that code the same changes as the failing PO r_{exe}, which have been accumulated in the PO set:

$$\mathscr{R}_{exe} = \{r_i | a_i = a_{exe}, e_i = e_{exe}\}, \tag{15}$$

where a_{exe} and e_{exe} are the action and effect parts of r_{exe}, respectively. The POs in $\mathscr{R}_{exe}$ are those generated during the learning process that represent more evidence of producing the coded effect, i.e., those which have a higher probability (see Eq. (10)) of producing the expected effect when applied [2]. The key idea is to use the POs in $\mathscr{R}_{exe}$ to search for new combinations of potentially relevant descriptors. The number of POs in $\mathscr{R}_{exe}$ might be quite large, mainly during the later stages of learning, so we select n for refinement to control the computational effort required. These n POs are drawn randomly according to the probability in Eq. (10), by defining the set $\mathscr{R}_n \subset \mathscr{R}_{exe}$.

Refining only the POs in $\mathscr{R}_n$ does not ensure that all possible combinations of descriptors have a chance of occurring, which is required to find all possible combinations of relevant descriptors. This is because the POs in $\mathscr{R}_n$ only test combinations of descriptors in a limited number of regions of the state space and further refinements will only continue searching in those specific regions. This is convenient for limiting the search, but the uncertainties related to the limited amount of samples demand some exploration in other regions of the state space. This is achieved by expanding the set $\mathscr{R}_n$ as:

$$\mathscr{R}_{comb} = \mathscr{R}_n \cup \mathscr{R}_{basic}, \tag{16}$$

where $\mathscr{R}_{basic}$ is a set of POs with preconditions that include a single descriptor in addition to those that should change with the action, i.e.,

$$\mathscr{R}_{basic} = \{r_j | p_j = \{p_{change}, d_j\}, a_j = a_{exe}, e_j = e_{exe}\}, \tag{17}$$

[2] As shown at the end of this section, this set of POs is formed by accumulating the m POs stored in the memory after the refinement process.

where $p_{change} \subseteq p_{exe}$ is the set of descriptors that should change when the PO is executed, and d_j, $j = 1, ..., |D^+|$ is a descriptor from the list D^+, which contains the individual descriptors observed in the states before the execution of the action in the positive instances (5). Effectively, the POs in $\mathscr{R}_{basic}$ are seeds used in other regions of the state space to meet the exploration requirements. It should be noted that we only consider descriptors from the positive instances, thereby prevent the exploration of regions where there is no evidence that they might contain relevant descriptors.

After the set $\mathscr{R}_{comb}$ is formed, we generate a new set of POs and their preconditions are obtained from all of the possible combinations of the preconditions of the POs in $\mathscr{R}_{comb}$. This defines the set of *candidate* POs,

$$\mathscr{R}_{cand} = \{r_k | p_k = \{p_i \cup p_j\}, a_k = a_{exe}, e_k = e_{exe}\}, \tag{18}$$

where p_i and p_j are the preconditions of any two POs in $\mathscr{R}_{comb}$. In this manner, we generate two different refinements of each PO $r_i \in \mathscr{R}_{comb}$. The first involves generating all the specializations in one descriptor from the list D^+,

$$\mathscr{R}_i^{[1]} = \{r_j | p_j = \{p_i, d_k\}, a_j = a_i, e_j = e_i\}, \tag{19}$$

where $\mathscr{R}_i^{[1]}$ is the set of POs generated from specializing r_i in one descriptor d_k from the list D^+; and p_i, a_i, and e_i are the precondition, action, and effect parts of r_i, respectively.

The second refinement method involves specializing the precondition of r_i using the descriptors of the preconditions of the other POs in $\mathscr{R}_{comb}$,

$$\mathscr{R}_i^{[2]} = \{r_j | p_j = \{p_i \cup p_k\}, a_j = a_i, e_j = e_i\}, \tag{20}$$

where $\mathscr{R}_i^{[2]}$ is the set of POs generated from this refinement and p_k is the precondition of any other PO $r_k \in \mathscr{R}_{comb}$. Refinement in this manner allows more than one descriptor to be tested each time, which increases the likelihood of finding the set of relevant descriptors. Note that the random selection of the POs in $\mathscr{R}_n$ according to their probability favors the combination of descriptors that represent more evidence to produce the coded changes, while also allowing combinations of descriptors that might not have a high probability when considered separately, but which may actually have a high probability when combined.

From the set of candidate POs, that with the highest probability, which is named as the *winner* PO r_w, is selected to replace the failing PO r_{exe}, where

$$w = \underset{j \in \mathbf{i}_{cand}}{\operatorname{argmax}} P_j, \tag{21}$$

$\mathbf{i}_{cand}$ is the set of indexes for the POs in $\mathscr{R}_{cand}$ and P_j is the probability for PO $r_j \in \mathscr{R}_{cand}$.

Finally, instead of discarding all the other POs in $\mathscr{R}_{cand}$, we store the m most accurate POs, $\mathscr{R}_m \subset \mathscr{R}_{cand}$, in memory for future refinements.

4.3. PO Elimination

To control the proliferation of POs, we eliminate those that satisfy the conditions

$$P_i < P_{thr} \tag{22}$$

and

$$\frac{n_i^+ + n_i^-}{n_i^T} > N_{thr}, \tag{23}$$

where P_{thr} is a threshold defined for the PO probability, P_i, and N_{thr} is a threshold for the normalized number of instances covered by the PO. The first criterion selects the POs with low probabilities of obtaining the coded changes and the second criterion prevents the elimination of the POs with too few instances to produce confident estimates. Note that using a probabilistic approach to evaluate the operators allows all the operators that code non-causative changes to have very low probabilities. Eventually, these operators will be removed during the elimination process, thereby preventing the planner from using them for planning.

After a specific number of iterations, it_{elim}, we eliminate all the POs that satisfy the elimination criteria.

4.4. The Learning Algorithm

The processes required for learning the POs are summarized in Algorithm 2. These processes replace the LEARNER function in the iDMF algorithm (Algorithm 1).

Algorithm 2 $\mathscr{R} \leftarrow$ LEARNER$(\mathscr{R}, I_{t-1})$

```
Update probabilities $P_i$ of POs in $\mathscr{R}$ using instance $I_{t-1}$
Get action $a_{t-1}$ from $I_{t-1}$
if $a_{t-1}$ was instructed then
    Generate $r_{inst}$ {PO generated from an action instruction (Section 4.2.1)}
    Set $r_{inst}$ as available for planning
    $\mathscr{R} \leftarrow \{\mathscr{R}, r_{inst}\}$
else
    Get executed PO $r_{exe}$ from $\mathscr{R}$
    Get state $s_t$ from $I_{t-1}$
    if $e_{exe} \not\subseteq s_t$ then
        Generate $\mathscr{R}_{cand}$ {candidate POs for $r_{exe}$}
        Get most accurate POs $\mathscr{R}_m \subset \mathscr{R}_{cand}$
        $w = \underset{j \in \mathbf{i}_{cand}}{\operatorname{argmax}} P_j$ {get index of the winner PO}
        Set $r_w$ as available for planning
        Set $r_{exe}$ as not available for planning
        $\mathscr{R} \leftarrow \{\mathscr{R} \cup \mathscr{R}_m\}$
    end if
end if
if ($t$ modulo $it_{elim}$)=1 then
    Eliminate POs that satisfy the elimination criteria {Section 4.3}
end if
```

5. Performance Evaluation

To evaluate the iDMF, we selected the game Sokoban [6] as a benchmark application. The Sokoban game scenario comprises a grid world where objects are placed in different cells. Given a specific goal that comprises the desired destination cells of the target objects, the agent should learn to move the target objects to the specified positions using vertical or horizontal movements. To reach the goal, the agent may be forced to move in an ordered manner objects blocking the trajectories of the target objects.

There are two reasons for using the Sokoban game as a test bed. First, many human-like tasks require moving objects in a specific order using actions such as pushing, picking, and placing, as found in the Sokoban game. For example, the task of removing a specific medicine bottle from a shelf may require pushing other bottles in an ordered manner, depending on the free spaces, to open a path to the required medicine. The same task definition can be used to sort perishable food on a shelf according to the expiration date, where the food with an earlier expiration date is placed at the front to ensure its more rapid consumption. Using a pick-and-place action, the robot can also order the parts of an object before their assembly on an industrial belt conveyor, or even set a table, which requires the consideration of plates, glasses, and cutlery in an ordered manner to satisfy the free spaces. The second reason for using the Sokoban game is that it is a simple but still challenging benchmark, which is used widely to evaluate logic-based planning methods [9, 14]. This facilitates a clearer evaluation of the performance of our method when learning POs. In addition, the Sokoban game provides a clear visualization of interesting cases where there are synergies between the planner and the learner, and the actions can be instructed easily by a lay person.

The notation used in the planning problem definition is as follows. Each cell may contain the target object, *to*, a non-target object, *o*, or be empty, *em*. The position of a cell is described by its coordinates in the grid with respect to the goal cell using a $(row, column)$ notation. The rows are counted downward when the cell is below the goal cell and upward otherwise. Positive counts for the columns are for cells to the right of the goal cell, and negative otherwise. The situation of a cell is described using predicates of the form $object(row, column)$, where *object* refers to the content of the cell and $(row, column)$ is the cell position. For instance, the predicate $to(-3,1)$ is true when the target object is three rows below and one column to the right of the goal cell. In this notation, the goal is always specified as $g = to(0,0)$. An action involves moving an object using horizontal or vertical movements. To describe an action, we use the notation: $move(pos, dir, ncells)$, where $pos = (row, column)$ is the position of the object that needs to be moved, $dir \in \{\text{UP, DOWN, LEFT, RIGHT}\}$ is the direction of movement, and *ncells* is the number of cells that the object will be moved.

To assess the validity and scalability of the proposed approach, we performed evaluations in simulated and real environments with different levels of complexity. In all cases, we used the PKS planner [19], which is known to be an efficient logic-based planner when integrated in the proposed framework. During processing, no POs were eliminated to facilitate a better assessment of the PO generation rate.

5.1. Performance Evaluation in a Simulated Environment

The evaluation was performed in a simulated environment using different domain sizes, each of which contained 11 randomly placed objects in cells. Figure 3 shows an example of a domain with a 5×5 grid. We considered several objects to increase the complexity of the scenario and to allow the learning of interesting POs.

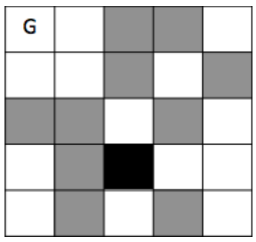
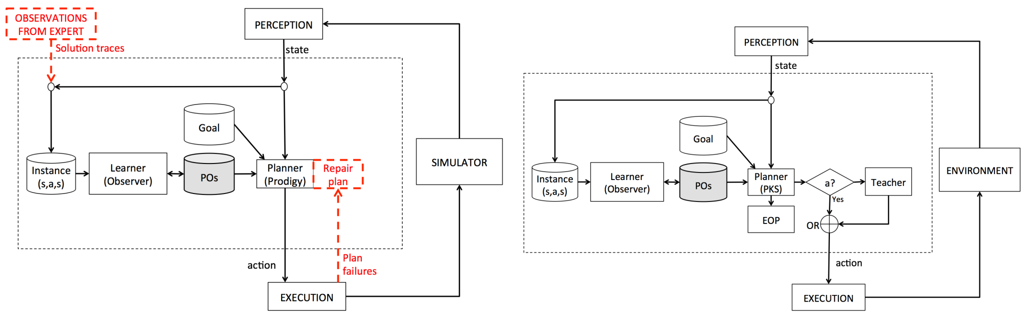


Figure 3: Sokoban scenario. The black cell represents the target object and other objects are shown by gray cells. Empty cells are white. The letter G denotes the goal position.

We conducted four different evaluations using our approach. First, in order to obtain a performance reference, we compared the results obtained by our learning approach with those obtained using the OBSERVER method [27, 28] when learning POs. This method was designed to be interleaved with planning to learn operators via a "learning by doing" approach, where instructions from a human are used to generate POs, which are subsequently refined based on experience in a simulated environment. This approach is an adaptation of the version space method [17], which for each PO, retains and refines the most specific and more general representation of the precondition of the PO. Similar to our method, a PO is generated after an action instruction, where the most general representation is generated from the observed changes and the most specific representation is initialized to the state before the execution of the action. The most general representation of the precondition of the POs is used for planning with a modified PRODIGY planner [25]. If a PO produces no change after its execution, the planner *repairs* the plan using the most specific representation to determine whether other conditions should be met in order to allow the coded changes to occur. It is important to note that repairing the plan does not modify the existing POs but it aims to find other POs, for which the requested conditions for the plan continuation are obtained in the resulting state when they are applied. We did not use the modified PRODIGY planner in the evaluation, so plan repair was not considered in the comparison. Another difference from the original framework employed by OBSERVER is that no solution traces are provided in advance in our method. Figure 4 shows the main differences between the original OBSERVER framework and the application of OBSERVER to our framework.

In our second experimental evaluation of the iDMF, we compared the results obtained by applying our approach to two different cases: starting with a set of previously accumulated experiences and starting with no previous experiences. We performed this experiment to determine how our approach behaves when the teacher is also allowed to provide solution traces to the goal in advance as well as during runtime, which is a method employed by other approaches for learning POs [27, 28].



(a) OBSERVER in its original framework.

(b) OBSERVER in our framework.

Figure 4: Schemas of the original framework using OBSERVER [27], which has been rearranged for convenience, and of our iDMF using OBSERVER. The modules of the original framework for plan repair and for the generation of the expert's solution traces, which are not considered in our iDMF, are denoted by dashed red lines (see Fig. 4(a)).

In the third experiment, we assessed the scalability of our approach, where we compared the results obtained in scenarios with increasing complexity: 5×5, 6×6, and 7×7 grid worlds. Note that increasing the size of the grid considerably increases the size of the search space as well as the number of relevant descriptors that need to be found to perform the task successfully.

Finally, we evaluated the sensitivity of the system to the learning parameters, n and m, which regulate the number of POs generated in parallel, and thus the speed of learning (Section 4.2.2). Therefore, we defined a scenario that required learning POs with several non-changeable relevant descriptors to compare the speed at which these descriptors were found using different values for these parameters.

In all the evaluations, we placed the goal cell in the upper left corner of the grid and we generated planning problems by placing the objects in random positions. The improvement in the decision-making performance was measured based on the ratio of teacher interventions during the learning process, the ratio of unexpected effects obtained during action executions, the ratio of plans completed successfully (without teacher interventions or unexpected effects), and the accumulated number of POs generated during the system runs. The parameters selected for the evaluation and generation of POs (see Section 4.2) were $c = 0.5$ and $n^T = 50$. The parameters n and m were set to 2 and 1, respectively, as well as other values in the last experiment, where we analyzed the sensitivity to these parameters. In the evaluation, we averaged the results of 10 runs, and we visualized the mean and standard deviation values.

5.1.1. Results

Figure 5 presents the average results of runs of 400 planning problems each, obtained with our learning approach (continuous blue line), and those obtained with the OBSERVER method (dashed red line). In general, our results demonstrate that the iDMF improves the decision-making capacity while learning progresses. At the beginning, the system required many teacher instructions due to missing POs, which prevented the planner from finding a plan (Fig. 5(a)). The number of teacher interventions decreased as learning proceeded until the system became fully autonomous, at which point help from the teacher was no longer required.

Unexpected effects (Fig. 5(b)) started appearing after some POs were generated from instructions. This is expected because at the very beginning of the learning, the PO set is almost empty and some POs need to be generated before the planner can use them for planning. These new POs, which could have unexpected results, caused an increase in the unexpected effects. Figure 5(b) shows that our learning approach generates POs that produce fewer unexpected effects than the OBSERVER. This is because our learning method is much faster at finding the relevant preconditions that allow the changes coded in the PO to be determined. This is also reflected in Fig. 5(d), which shows the ratio of successful plans, i.e., those completed without instructions or unexpected effects. This figure shows that the percentage of plans completed successfully using our approach was higher than that using OBSERVER at all stages of learning. Our method completed 80 % of the plans successfully in the early stages.

The total number of POs generated with our approach was higher than the number of POs generated with OBSERVER. This is because our method memorizes many POs that encode the same changes but with many alternative preconditions, whereas OBSERVER only evaluates two preconditions for each change: the most general and the most specific. The number of instructions required during learning was similar in both cases (Fig. 5(a)). We should note that not all of the teacher interventions initiated a new PO because the planner used for the experiments did not generate a plan if at least one missing PO prevented a complete plan from being found. For instance, if the generation of a plan was not possible because only one PO was missing, some of the instructed actions produced pre-existing POs until the missing PO was generated and added to the PO set. However, although the lack of information about which PO is missing demands more effort from the teacher, the number of instructions decreases rapidly to zero as learning progresses. This is because the iDMF searches for a new plan at each iteration by replanning (see Section 3), which allows the planner to continue with the task immediately as soon as the missing operator is instructed.

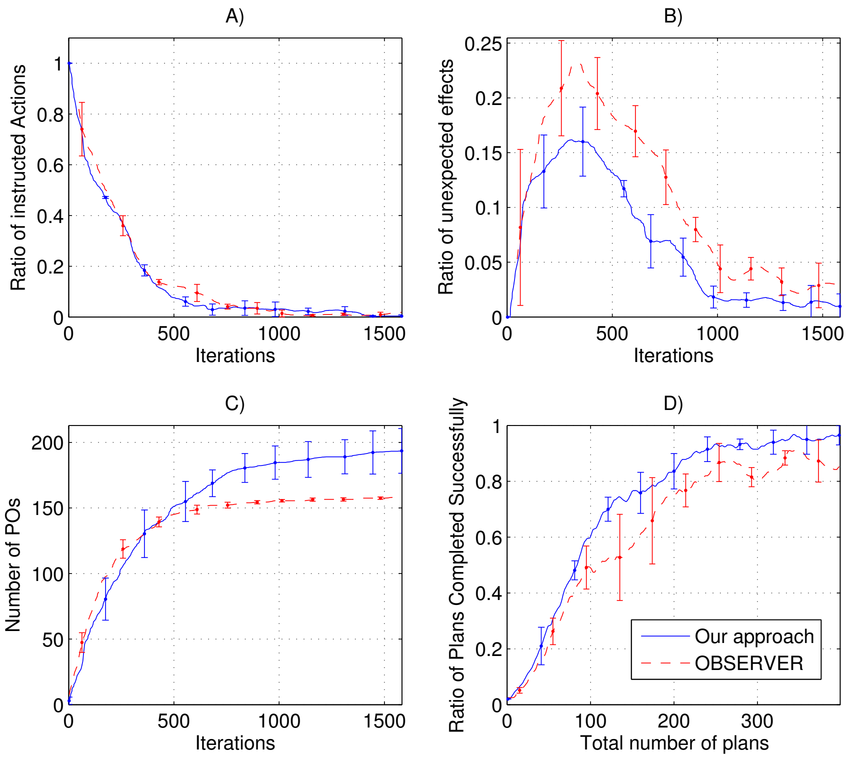


Figure 5: Average of 10 runs of 400 planning problems. The results obtained using our learning approach are denoted by the continuous blue line, whereas those obtained using the OBSERVER method are shown by the dashed red line. A) Ratio of teacher interventions during learning. B) Ratio of unexpected effects during learning. C) Accumulated number of POs generated during learning. D) Ratio of plans completed successfully (without teacher instructions or unexpected effects).

To provide an insight into the structure of POs, Fig. 6 shows a graphical description of a complete sequence of states and operators obtained after executing a plan. In the figure, each column corresponds to a step in the plan. The state, the executed PO, and the changed descriptors are shown for each step. The initial state of the sequence is

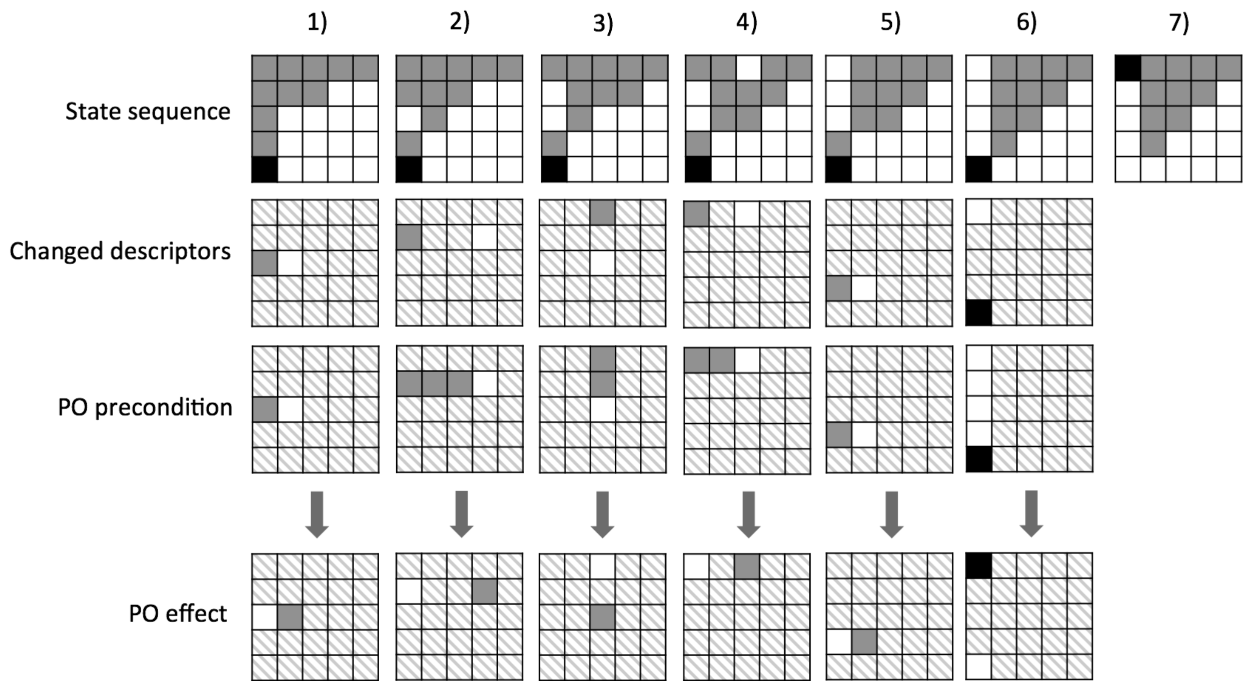


Figure 6: Example of a plan generated by the PKS planner. Dashed cells represent the cells that were not considered in the precondition or effect parts of the PO.

$$\begin{aligned} s_1 \quad = \{ \quad & o(0,0), o(0,1), o(0,2), o(0,3), o(0,4), \\ & o(-1,0), o(-1,1), o(-1,2), em(-1,3), em(-1,4), \\ & o(-2,0), em(-2,1), em(-2,2), em(-2,3), em(-2,4), \\ & o(-3,0), em(-3,1), em(-3,2), em(-3,3), em(-3,4), \\ & to(-4,0), em(-4,1), em(-4,2), em(-4,3), em(-4,4)\}, \end{aligned}$$

while the PO at the first step is

$$\begin{aligned} & p = \{o(-2,0), em(-2,1)\}, \\ & a = move((-2,0), \text{RIGHT}, 1), \\ & e = \{em(-2,0), o(-2,1)\}. \end{aligned}$$

As shown in the figure, steps one and five involve POs that only consider descriptors changed by the action. These descriptors are sufficient to obtain the changes because the actions only involve moving an object to an adjacent cell, and thus considering the situations of the initial and final cells in the precondition is sufficient to allow the changes. Therefore, these POs did not produce any unexpected effects after their creation from an action instruction, so they required no further refinements. This was not the case for the other steps in the plan, e.g., in step three, the precondition of the PO needed to be refined with the unchanged descriptor $o(-1,2)$ in order to obtain the coded changes,

$$\begin{aligned} & p = \{o(0,2), em(-2,2), o(-1,2)\}, \\ & a = move((0,2), \text{DOWN}, 1), \\ & e = \{em(0,2), o(-2,2)\}, \end{aligned}$$

because the change from an empty cell, *em*, to an occupied cell, *o*, for a cell two positions down from the pushed object using one-cell pushing was only possible if another object was pushed in between them. This was ensured by considering $o(-1,2)$ in the precondition of the PO. A similar case occurs in step four. In step two, the PO used required a larger refinement because two unchanged descriptors, rather than one, needed to be considered to obtain a change three cells further in the direction of the movement.

In step six, a PO that required several refinements was applied,

$$\begin{aligned}&p = \{to(-4,0), em(0,0), em(-3,0), em(-2,0), em(-1,0)\},\\ &a = move((-4,0), \mathrm{UP}, 4),\\ &e = \{em(0,2), o(-2,2)\}.\end{aligned} \tag{24}$$

In this case, the action involves moving the target object several cells up to the goal position. The initial version of the PO, which was created after an action instruction, only considered the changed descriptors $to(-4,0)$ and $em(0,0)$ in its precondition. However, to make these changes, all of the cells between the initial and final cells should be empty. Since these cells were not considered originally, many unexpected effects occurred. After the refinements, the precondition of the PO also included the unchanged descriptors $em(-3,0)$, $em(-2,0)$, and $em(-1,0)$, which actually ensured a free path between the initial and final cells.

The results of the experiments that considered previously accumulated instances are presented in Fig. 7. In this experiment, we compared the results obtained with our approach starting with no previous instances, as well as with an initial set of instances accumulated before learning began. In the latter case, we used the set of instances accumulated from the 400 planning problems in one of the runs of the experiments shown in Fig. 5. The results show that the performance of the system was much better when considering an initial set of instances compared with that using no previous examples. This is because the evaluation of the generated POs was more accurate from the very beginning as the calculation of the probability to obtain the expected effect was based on many instances (see Section 4.1). This facilitated the more rapid identification of those POs containing relevant descriptors in their preconditions (see Section 4.2.2), thereby increasing the speed of learning.

In order to evaluate the scalability of our approach with more complex applications, we performed two new experiments in more complex scenarios with 6×6 and 7×7 grid worlds. In these experiments, each run tested 800 planning problems to evaluate the convergence in more complex situations. The results are shown in Fig. 8, which demonstrate that the system could learn successfully in all the cases, where it became more fully autonomous and the number of unexpected effects declined as learning proceeded.

To demonstrate the capacity of our system for finding the relevant descriptors in much larger state spaces, we present two example of the POs generated during learning. The first is presented in (26) and it corresponds to the 6×6 grid case, where the target counter was required to move from the upper right corner to the goal cell at the upper left corner. A comparison of this PO with the initial version generated after action instruction (25) shows that the system could find the four unchanging causative relevant descriptors that produced the changes coded in the PO, i.e., the descriptors indicating that all the cells in the trajectory to the goal should be empty. The second example shows a PO generated for a 7×7 grid (see (28)) for moving the target counter from the lower left corner of the grid to the goal position. In this case, the number of unchangeable relevant descriptors was five, as shown by comparing this PO with its initial version (27) generated after action instruction.

$$\begin{aligned}&p = \{to(0,5), em(0,0)\},\\ &a = move((0,5), \mathrm{LEFT}, 5),\\ &e = \{em(0,5), to(0,0)\}.\end{aligned} \tag{25}$$

$$\begin{aligned}&p = \{to(0,5), em(0,4), em(0,3), em(0,2), em(0,1), em(0,0)\},\\ &a = move((0,5), \mathrm{LEFT}, 5),\\ &e = \{em(0,5), to(0,0)\}.\end{aligned} \tag{26}$$

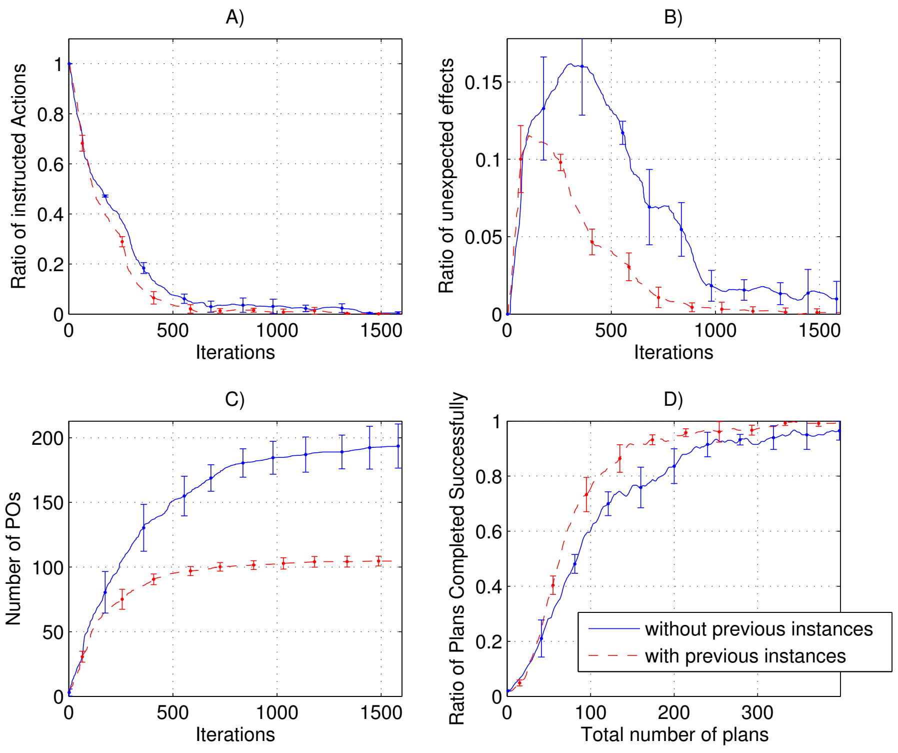


Figure 7: Comparison of the performance with and without using previously accumulated experiences.

$$\begin{aligned} p &= \{to(-6,0), em(0,0)\}, \\ a &= move((-6,0), \mathrm{UP}, 6), \\ e &= \{em(-6,0), to(0,0)\}. \end{aligned} \tag{27}$$

$$\begin{aligned} p &= \{to(-6,0), em(-5,0), em(-4,0), em(-3,0), em(-2,0), em(-1,0), em(0,0)\}, \\ a &= move((-6,0), \mathrm{UP}, 6), \\ e &= \{em(-6,0), to(0,0)\}. \end{aligned} \tag{28}$$

The results obtained in these scenarios with different degrees of complexity indicate that the system is scalable to more complex applications. In particular, the ratio of action instructions did not vary greatly in the different cases (Fig. 8(a)), thereby demonstrating that the teacher effort scaled very well with the complexity of the problem.

Finally, to evaluate the sensitivity of the learning approach to the parameters n and m, we used a scenario designed for this purpose, which comprised a 3×12 grid world (Fig. 9), where the target object was always placed in the fixed upper right corner position of the grid. All of the other objects were placed randomly, as before. The task involved moving the target counter to the goal position at the upper left corner of the grid using a single action. This implies that learning should generate a PO with 10 unchangeable relevant descriptors in its precondition (those indicating that all the cells from the target object to the goal should be empty) to complete the task successfully. This learning problem is not trivial because there are many other irrelevant descriptors in the grid, which should be distinguished

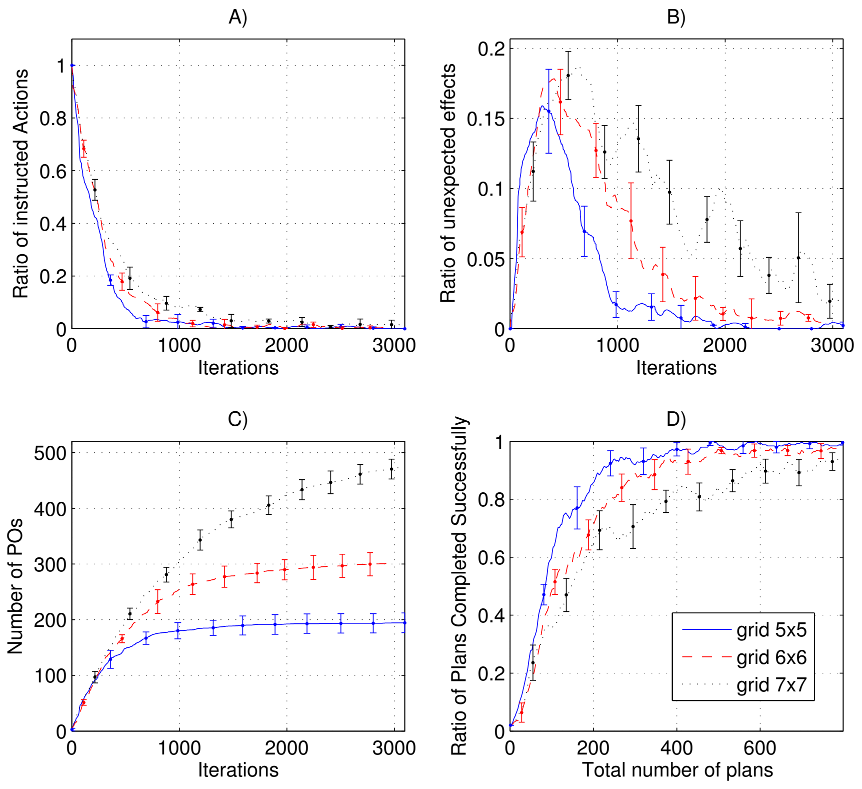


Figure 8: Comparison of the performance with 5×5, 6×6, and 7×7 grid worlds.

from the relevant descriptors. To clearly evaluate the learning performance, we specified the problem so only the aforementioned PO could produce unexpected effects by restricting the actions of all the other POs to moving objects only to adjacent cells and without pushing other objects. This prevented the other POs from producing unexpected effects when they were applied because all the relevant descriptors were those that changed with the action. For comparison, we set the values of n and m to 2 and 1; 4 and 1; and 4 and 2, respectively.

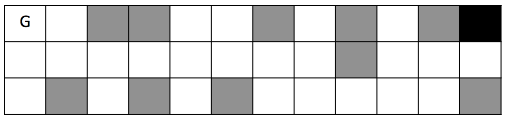


Figure 9: Example of a Sokoban scenario in a 3×12 grid world. The black cell represents the target object and the other objects are represented by gray cells. Empty cells are shown in white. The letter G denotes the goal position.

Figure 10 presents the average results of 10 runs, each of them comprising 50 planning problems. Figure 10(b) shows that the ratio of unexpected effects was smaller for higher values of these parameters. It should be noted that the improvements shown for the case where $n = 4$, $m = 1$ indicate that increasing only the value of n affected the learning speed. However, increasing only the value of m may also accelerate learning, as indicated by the improvements achieved when m was increased from 1 to 2 while keeping $n = 4$. The best results correspond to the values of $n = 4$, $m = 2$, for which the ratio of unexpected effects dropped to zero rapidly.

We note that the ratio of successfully completed plans was similar in all cases, which may be explained by the fact that a plan was considered unsuccessful if there was at least one instruction or unexpected effect. This prevented us from determining the exact number of unexpected effects that occurred while a plan was being executed, and thus the actual learning performance. As expected, the accumulated number of POs generated during learning (Fig. 10(c)) was greater for higher values of m because this parameter represents the number of POs stored in memory for future refinements (see Section 4.2.2).

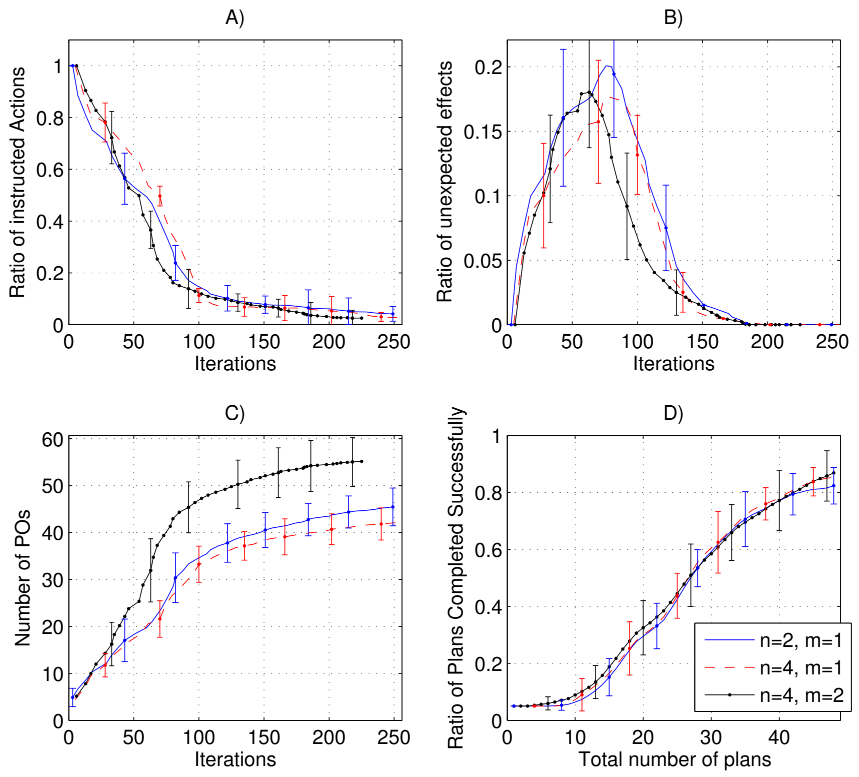


Figure 10: Comparison of the performance when using different values of the parameters n and m in the 3×12 grid world scenario.

To obtain further insights into the effects of changing n and m, Fig. 11 shows the accumulated number of unexpected effects throughout the learning process, which demonstrates that the accumulation of unexpected effects was comparably lower for larger values of these parameters. In particular, the average numbers of unexpected effects after learning were 16, 14, and 13 for the cases where: $n = 2$, $m = 1$; $n = 4$, $m = 1$; and $n = 4$, $m = 2$, respectively. In all cases, the final PO generated was

$$\begin{aligned}
&p = \{to(0,11), em(0,10), em(0,9), em(0,8), em(0,7), em(0,6),\\
&em(0,5), em(0,4), em(0,3), em(0,2), em(0,1), em(0,0)\},\\
&a = move((0,11), \text{LEFT}, 11),\\
&e = \{em(0,11), to(0,0)\}.
\end{aligned} \tag{29}$$

Finally, we performed experiments to evaluate the sensitivity to the parameter c, i.e., for $c = 0.4$ and $c = 0.6$. We detected a low sensitivity to this parameter. The learning profile was similar for all of the values tested and the system successfully completed 80% of the plan in approximately the same number of plans (see Fig. 5(d) as a reference). This low sensitivity may be attributable to the small bias introduced by considering values of c close to 0.5. These

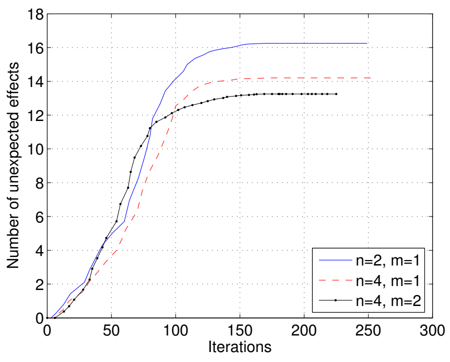


Figure 11: Comparison of the accumulated number of unexpected effects during the learning process for different values of n and m.

values still allow the learning approach to correctly distinguish between refinements of a PO that would lead to higher or lower probabilities of obtaining the effect. The low sensitivity to this parameter is clearly beneficial.

5.2. *Performance Evaluation on Robot Platforms*

In order to test the feasibility and scalability of the proposed iDMF in different real robot platforms and to demonstrate the synergies between the integrated components, we implemented the task of moving and arranging objects in an ordered manner using two different robot platforms: the Stäubli arm and the humanoid robot ARMAR III [3].

5.2.1. *The Stäubli Robot Arm*

An example scenario of the Sokoban task implemented using the Stäubli arm is shown in Fig. 12. In this application, the robot moved black counters in an ordered manner so the target counter, having a red mark, reached a given goal position.

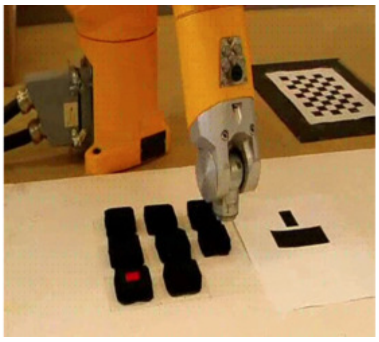


Figure 12: Scenario showing the Sokoban task with the Stäubli arm platform in a 3×3 grid world.

To build the symbolic state representation, we generated a virtual grid, which divided the image of the scenario captured after each planning-learning loop into cells. We then measured the average grayscale of the pixels in a cell and sent it to a Bayesian classifier [13], which classified each cell as either empty or containing an object. A grayscale value closer to white indicated that the cell was empty, whereas a value closer to black denoted that an object occupied the cell. To identify the target object, we kept track of its movements. The Bayesian classifier was trained during runtime using samples provided by the teacher, who supervised the classification process at each iteration. If misclassification occurred, the teacher indicated the correct classes and the parameters of the classifier were updated. All of these processes occurred in the perception module (see Fig. 1).

The state description was sent to the iDMF, which performed a learning step and a planning step, thereby yielding the next action to execute. If the planner could not find a plan, the teacher was presented with an interface so he could specify the next cell where the object should be placed, the direction of movement, and the number of steps.

The action provided by the planner or the teacher was sent to the action module for its actual execution. The module decomposed the action into four parts, each of which comprised the initial and final positions of the end-effector in the Cartesian space. These coordinates were sent in sequence to the controller of the robot for action execution. First, the robot moved the end-effector from the standby position at the side of the grid to the vertical axis of the center of the cell containing the counter to be moved. Next, the end-effector was moved down to a predefined height with respect to the table. This height was defined so the end-effector pressed the counter with sufficient force to allow the smooth displacement of the counter on the table. The end-effector was then moved in a straight line in the horizontal plane to the center of the final cell. Finally, the robot released the counter and returned to the standby position.

Experiments

Two different experiments were performed using the Stäubli arm platform. The first was implemented in a scenario that comprised a 3×3 grid world with eight black counters placed in the cells (Fig. 12). This configuration is particularly interesting because it allows the definition of planning problems that may require several actions.

In this experiment, many POs were generated during the learning process involving one-cell actions due to the movement restrictions when all the cells were occupied except one. Although these POs were simpler than those with multi-cell actions (e.g., (24)), some of them still required refinements due to unexpected effects (Section 4.2.2). This was the case for those POs that pushed two counters at the same time, such as the first PO executed in the sequence shown in Fig. 14,

$$\begin{aligned} p &= \{em(0,-2), o(0,-1), o(0,0)\}, \\ a &= move((0,0), \text{LEFT}, 1), \\ e &= \{o(0,-2), em(0,0)\}. \end{aligned} \tag{30}$$

Note that the precondition of PO (30) contains the descriptor $o(0,-1)$, which did not change with the action but it was relevant for obtaining the coded effect.

The robot was trained in this scenario using different positions for the target counter and the goal. After learning, the numbers of accumulated instructions, unexpected effects, and POs were 22, three, and 25, respectively. Using the learned POs, the robot was capable of solving difficult situations, such as that presented in Fig. 13. In this case, the target counter placed in the lower middle cell had to be moved to the upper right corner of the grid. The initial state for this problem was

$$\begin{aligned} s_{ini} = \{ \quad & em(0,-2), o(0,-1), o(0,0), \\ & o(-1,-2), o(-1,-1), o(-1,0), \\ & o(-2,-2), to(-2,-1), o(-2,0)\}. \end{aligned}$$

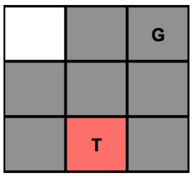


Figure 13: Graphical representation of the initial situation for an example task using the 3×3 grid world scenario. The target counter is red and labeled with the letter T. The goal cell is labeled with the letter G.

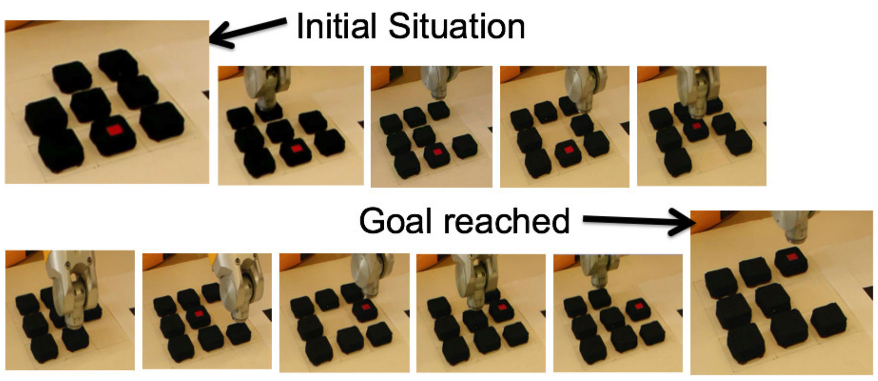


Figure 14: Snapshots showing the sequence of actions executed to move the target counter to the upper right position.

The complete execution of the task is presented in Fig. 14 (a video is available at https://dl.dropbox.com/u/19473422/STAEUBLI.wmv).

To better illustrate the generation of POs from unexpected effects (Section 4.2.2), we performed a second experiment in a scenario that used a 3×5 grid world, where a maximum of seven counters was placed on the grid (Fig. 15). There were many empty cells in this case, so it was possible for the teacher to instruct actions involving several steps, thereby leading to the generation of POs that required many refinements. To perform this experiment, we defined planning problems where the goal position was in the upper left corner of the grid but different initial situations were used to demonstrate how a PO is generated from instructions and refined from unexpected effects.

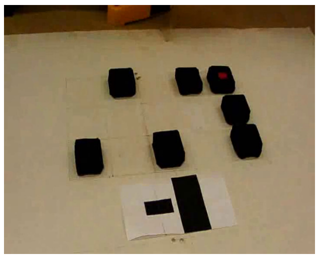


Figure 15: Scenario involving the Sokoban task using the Stäubli arm platform in a 3×5 grid world.

The first initial situation, which is shown in Fig. 16, is represented by the state

$$\begin{aligned} s_{ini} = \{ \; & em(0,0), em(0,1), em(0,2), em(0,3), to(0,4), \\ & em(-1,0), em(-1,1), em(-1,2), em(-1,3), o(-1,4), \\ & o(-2,0), em(-2,1), o(-2,2), em(-2,3), o(-2,4)\}. \end{aligned}$$

Because the experiment started with an empty PO set, the teacher was asked for an instruction. The action instructed by the teacher was $a = move((0,4), \text{LEFT}, 4)$, which yielded the state transition shown in Fig. 17(a) and 17(b). Based on this instruction, a PO was generated that only coded the observed change (see Section 4.2.1):

$$\begin{aligned} & p = \{to(0,4), em(0,0)\}, \\ & a = move((0,4), \text{LEFT}, 4), \\ & e = \{em(0,4), to(0,0)\}. \end{aligned} \tag{31}$$

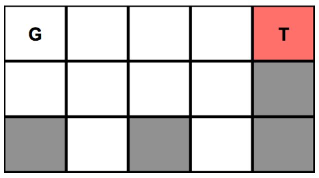


Figure 16: Graphical representation of the first initial situation in the example task using the 3×5 grid world scenario. The target counter is red and labeled with the letter T. The goal cell is labeled with the letter G.

Note that PO (31) only considers the descriptors at the upper right and left corners of the grid, but not those in the trajectory between them because they did not change with the action. Therefore, we can easily force an unexpected effect by defining a new initial situation where a counter is placed on the trajectory from the target counter to the goal (Fig. 18). This initial situation is described by the state

$$\begin{aligned} s_{ini} = \{ \; & em(0,0), o(0,1), em(0,2), em(0,3), to(0,4), \\ & em(-1,0), em(-1,1), em(-1,2), em(-1,3), o(-1,4), \\ & o(-2,0), em(-2,1), o(-2,2), em(-2,3), o(-2,4)\}. \end{aligned}$$

Given this initial situation and the PO set containing only the PO generated from the teacher instruction (31), the planner generated a plan that comprised this single PO. After executing the PO, the state transition experienced is shown in Fig. 17(c) and 17(d)[3]. The state describing the situation obtained is

$$\begin{aligned} s_{final} = \{ \; & o(0,0), to(0,1), em(0,2), em(0,3), em(0,4), \\ & em(-1,0), em(-1,1), em(-1,2), em(-1,3), o(-1,4), \\ & o(-2,0), em(-2,1), o(-2,2), em(-2,3), o(-2,4)\}. \end{aligned}$$

Since the expected effect coded by the PO was not included in the resulting state, i.e., $\{em(0,4), to(0,0)\} \not\subset s_{final}$, the mechanism related to generation from unexpected effects was triggered. Among all the candidate POs (18), the winner used to replace the failing PO (31) is

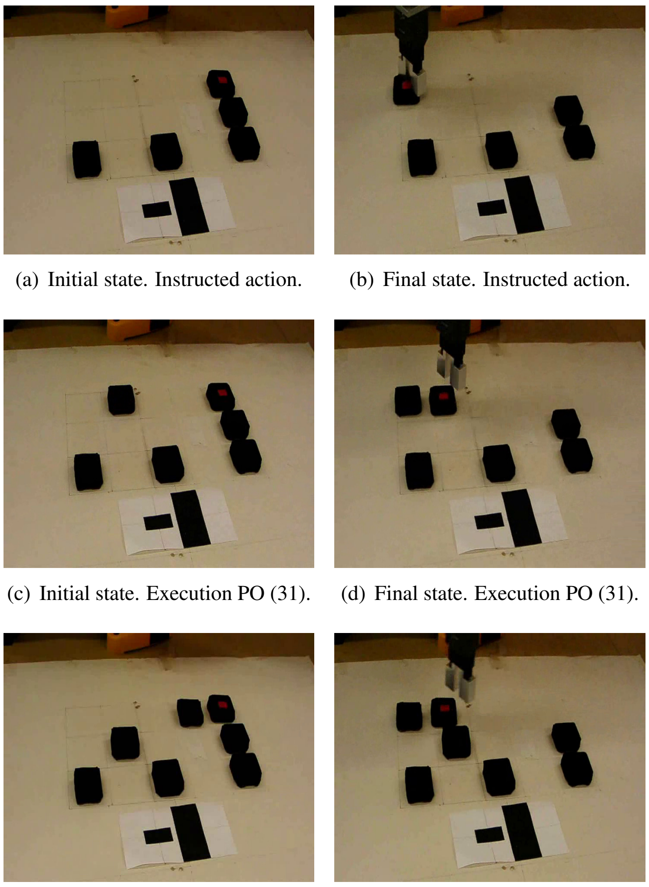
$$\begin{aligned} & p = \{to(0,4), em(0,1), em(0,0)\}, \\ & a = move((0,4), \text{LEFT}, 4), \\ & e = \{em(0,4), to(0,0)\}. \end{aligned} \tag{32}$$

Note that the precondition of the newly generated PO indicates that the cell where the blocking counter was placed initially should be empty in order to obtain the coded effect.

After the first refinement, we forced a second unexpected effect by placing a counter in a different cell on the trajectory to the goal. In this case, the initial state is (Fig. 19)

$$\begin{aligned} s_{ini} = \{ \; & em(0,0), em(0,1), em(0,2), o(0,3), to(0,4), \\ & em(-1,0), em(-1,1), em(-1,2), em(-1,3), o(-1,4), \\ & o(-2,0), em(-2,1), o(-2,2), em(-2,3), o(-2,4)\}. \end{aligned}$$

[3]Counters were not allowed to leave the grid, so the robot interrupted the execution of the action immediately when it detected that the continuation of the movement would move a counter outside the grid.



(a) Initial state. Instructed action.

(b) Final state. Instructed action.

(c) Initial state. Execution PO (31).

(d) Final state. Execution PO (31).

(e) Initial state. Execution PO (32).

(f) Final state. Execution PO (32).

Figure 17: Examples of state transitions.

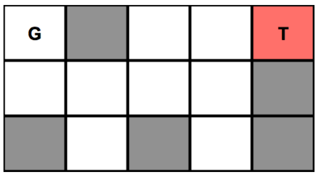


Figure 18: Graphical representation of the second initial situation in the example task using the 3×5 grid world scenario.

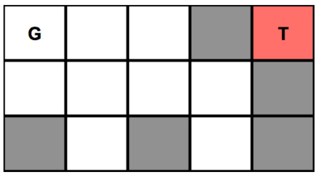


Figure 19: Graphical representation of the third initial situation in the example task using the 3×5 grid world scenario.

The planner generated a plan that involved the single PO (32) because it determined that it was suitable for the new initial situation. The execution of the PO produced the state transition shown in Fig. 17(e) and 17(f). As before, because the final state did not include the coded effect, the method for generating POs from unexpected effects was triggered and the failing PO was replaced with the new winner

$$\begin{aligned}
&p = \{to(0,4), em(0,3), em(0,1), em(0,0)\},\\
&a = move((0,4), \text{LEFT}, 4),\\
&e = \{em(0,4), to(0,0)\}.
\end{aligned} \tag{33}$$

The new PO included an additional descriptor in its precondition, which indicated that the cell of the initial state containing the new blocking counter should also be empty.

After the experiment was completed, the numbers of accumulated instructions, unexpected effects, and generated POs were five, two, and seven, respectively. To test the learned POs, we defined an initial state where the two blocking counters were placed at the same time (Fig. 20),

$$\begin{aligned}
s_{ini} \; = \{ \;& em(0,0), o(0,1), em(0,2), o(0,3), to(0,4),\\
& em(-1,0), em(-1,1), em(-1,2), em(-1,3), o(-1,4),\\
& o(-2,0), em(-2,1), o(-2,2), em(-2,3), o(-2,4)\}.
\end{aligned}$$

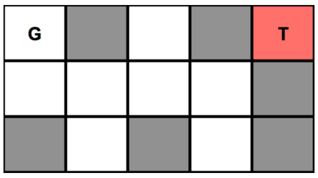


Figure 20: Graphical representation of the fourth initial situation in the example task using the 3×5 grid world scenario.

In this case, the planner could find a plan that involved applying two initial POs to free the path to the goal before executing the refined PO (32). The complete sequence is shown in Fig. 21.

A demo showing all the steps performed in this second experiment, is available at:
https://dl.dropbox.com/u/19473422/STAEUBLI2.wmv.

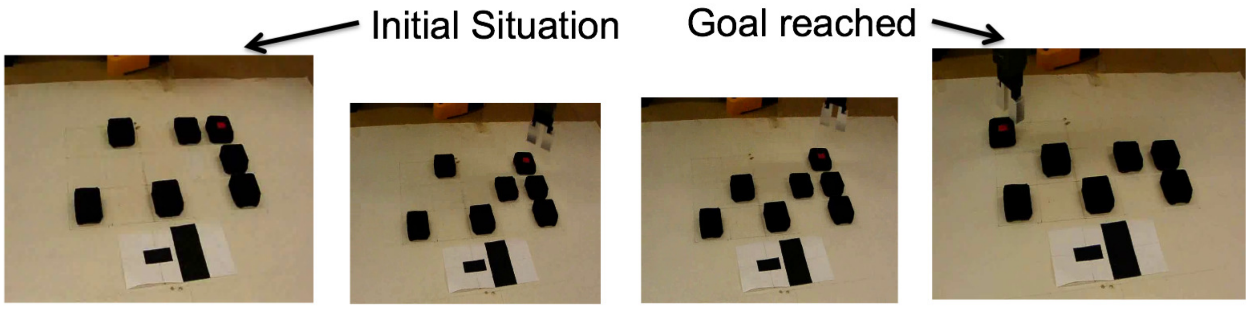


Figure 21: Example of a plan for moving the target counter from the upper right cell of the grid to the goal cell at the upper left corner when two counters blocked the trajectory.

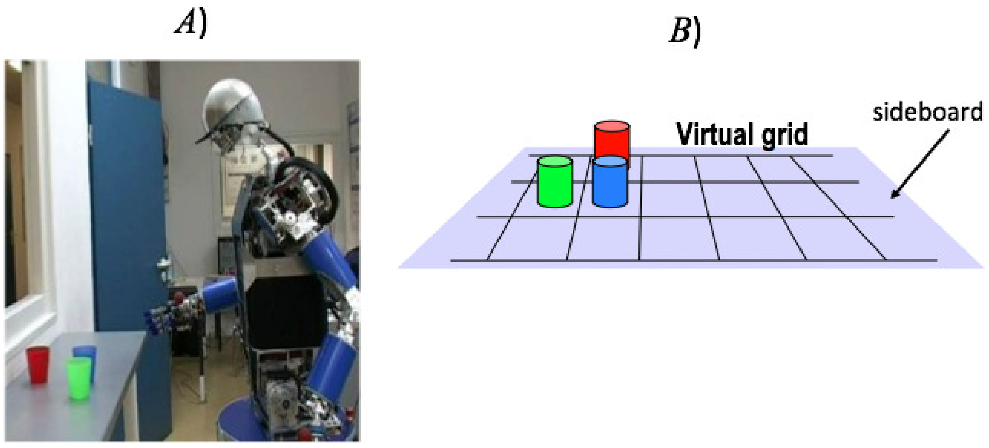


Figure 22: A) Scenario of the Sokoban task using cups as objects with the ARMAR III platform. B) Virtual grid generated to represent the problem as a Sokoban game.

5.2.2. ARMAR III Robot

The Sokoban game implemented using the ARMAR III platform involved moving a target cup on a sideboard where there were other blocking cups, but without colliding with them. Figure 22(a) shows a snapshot of this scenario.

To represent the positions of the cups in cells, we generated a virtual grid on the sideboard where each cup was placed inside a cell, as shown in Fig. 22(b). Each cup was recognized using methods implemented in the robot [3], which determined the positions of the centroids of the cups in the Cartesian plane parallel to the sideboard. A cup was considered to lie in the cell whose the center was at the shortest distance from the cup's centroid. The target cup was identified with a color set by the user (green in the example).

Actions were performed based on pick-and-place with grasping, where the robot was limited to performing simple straight movements of the cups in horizontal or vertical directions. First, the robot hand was placed in a graspable position on the cup that needed to be moved using visual servoing guided by the red ball in the wrist of the robot (see Fig. 22(a)). Next, the robot grasped the cup and lifted it vertically before performing a horizontal straight movement vertical to the center of the final cell. The robot then lowered the cup until it touched the sideboard, at which point the cup was released.

Experiments

This experiment involved moving a green cup (target cup) to the goal position in a grid that contained other blocking cups. After learning sufficient POs, the robot was capable of solving situations such as that presented in Fig. 23. In this case, the initial state is described by

$$\begin{aligned} s_{ini} \quad = \{ \quad & em(1,-2), o(1,-1), em(1,0), em(1,1), em(1,2), \\ & to(0,-2), o(0,-1), em(0,0), em(0,1), o(0,2), \\ & em(-1,-2), em(-1,-1), em(-1,0), em(-1,1), em(-1,2)\}. \end{aligned}$$

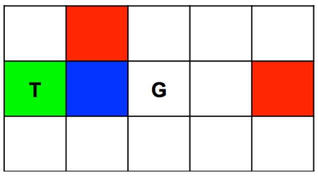


Figure 23: Graphical representation of the initial situation in the example task implemented using the ARMAR III platform. The letter T denotes the target cup and the letter G is the goal cell.

The trajectory from the target cup to the goal was blocked by a cup that could not be moved upward because another cup blocked this movement, while it could not be moved to the right because there was insufficient space for the robot's hand to release the cup without knocking over the cup furthest to the right. Using the learned POs, the ARMAR robot could generate a three-step plan to handle all of these restrictions, i.e., first moving the cup blocking the target cup one position to the right, where no cups blocked its upward movement, and then up. This freed the path of the target cup to reach the goal. The complete execution plan is presented in Fig. 24.

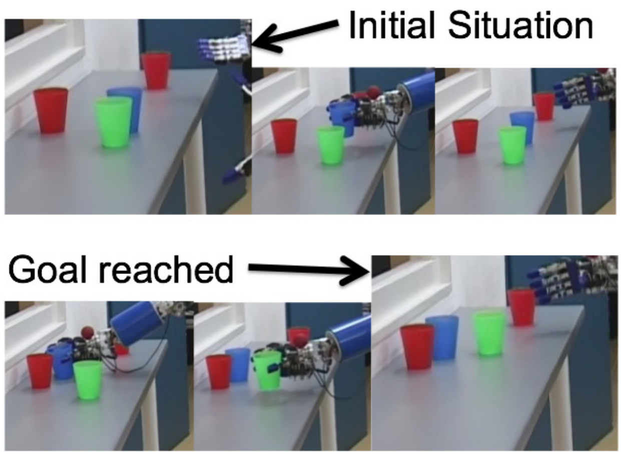


Figure 24: Example of the performance of the system in a complex situation with many blocking cups in the Sokoban task implemented using the ARMAR III platform.

The POs executed during each step of the plan were

$$\begin{aligned} & p = \{o(0,-1), em(0,0)\}, \\ & a = move((0,-1), \mathrm{RIGHT}, 1), \\ & e = \{em(0,-1), o(0,0)\}, \end{aligned} \tag{34}$$

$$\begin{aligned}
p &= \{o(0,0), em(1,0)\},\\
a &= move((0,0), \mathrm{UP}, 1),\\
e &= \{em(0,0), o(1,0)\},
\end{aligned} \tag{35}$$

and

$$\begin{aligned}
p &= \{to(0,-2), em(0,-1), em(0,0)\},\\
a &= move((0,-2), \mathrm{RIGHT}, 2),\\
e &= \{em(0,-2), to(0,0)\}.
\end{aligned} \tag{36}$$

In particular, the last PO (36), which moved the target cup two positions to the right, included the descriptor $em(0,-1)$ in its precondition, which did not change with the action. The addition of this descriptor to the precondition after an unexpected effect indicates that the cell in the middle of the trajectory to the goal should be empty to allow the successful execution of the PO. The numbers of instructions, unexpected effects, and generated POs throughout the experiment using this simple scenario were four, one, and five, respectively.

A demo of the executed task is available at: https://dl.dropbox.com/u/19473422/ARMAR.wmv.

6. Discussion

In this study, we proposed an iDMF that is suitable for robotic applications of human-like tasks. The iDMF rapidly learns how to execute tasks using an efficient learning approach, which produces useful POs with the help of a human teacher. The system becomes progressively more autonomous until teacher intervention is no longer required.

Two main factors facilitate this progression. First, AI techniques for reasoning and learning permit the use of a declarative knowledge representation, which is similar to human natural language. We used these techniques to develop an iDMF that facilitates the rapid learning of human-like tasks using simple and natural instructions from a human, instead of providing complex explanations of the task. Using our framework, a lay person can interact with a robot by providing simple action instructions, which rapidly improve the robot's performance while the task is executed because of a novel learning approach. This natural interaction is similar to that which occurs when one person instructs another.

The second factor that facilitated the integration of AI and robotics in our study is the fact that advanced robotics platforms can now perceive and manipulate objects found in human environments in a reliable manner. In this study, we used the ARMAR III humanoid robot, which can handle a wide variety of kitchen objects. Thus, perceiving and interacting with these objects facilitates the grounding of the POs generated by our approach. This is necessary to guarantee that a given PO does indeed yield the desired outcomes. As a consequence, our framework can integrate symbolic planning, learning, perceiving, and executing by a robot in a smooth and natural manner, which is one of the major challenges in the interface between AI and robotics. In this manner, we consider that our framework can facilitate the integration of robots into human society, at least by helping us with simple everyday chores.

Many elements can contribute to the rapid learning of POs. In the early stages of the learning, the most important role is that of the human teacher who guides the exploration of actions because the planner rarely makes a decision due to the excessive number of missing POs (see Section 4.2.1). In this case, the teacher instructs task-relevant actions, which lead to the generation of POs that become available immediately for planning. Thus, the planner is provided rapidly with task-relevant POs and after a few initial teacher interventions, they allow the system to generate and complete some plans autonomously (see Figs. 5(a) and 5(d)).

The POs generated from instructions allow plan generation in the early stages of the learning, but many of their executions have unexpected effects. This is because the POs generated from an instruction only will code observed changes, but none of the unchanging and possibly causative descriptors that also need to be considered to obtain a complete set of preconditions for the given PO. Consequently, the rate of unexpected effects surpasses the rate of required teacher interventions shortly after learning begins, as demonstrated in Figs. 5(a) and 5(b). The main learning role is then switched from the teacher to the mechanisms for PO generation from unexpected effects (Section 4.2.2). These mechanisms progressively refine the failing POs until no more unexpected effects occur (Fig. 5(b)).

Two complementary methods are used to solve unexpected effects rapidly. First, the method for PO evaluation using our *density*-estimate formula (Eq. (12)), where the *density*-estimate requires only a few samples to reliably evaluate different alternative POs that coded the same changes. The success of the *density*-estimate highlights the importance of considering the lack of experience in the probability estimate when estimation must be based on only a few experiences. The second method for rapidly solving unexpected effects involves searching for accurate POs using a parallel strategy. For each of the coded changes, the most accurate POs are memorized and refined in parallel using a strategy that assesses the preconditions of potentially relevant combinations of descriptors.

We compared our approach with a widely used approach for the online learning of POs, the OBSERVER [27]. The results demonstrate that our approach significantly outperforms OBSERVER because it can learn accurate POs with a much smaller number of samples. In addition, we successfully implemented and tested the iDMF using two very different robotic platforms, thereby demonstrating that our approach can be transferred.

The simplicity and efficiency of the iDMF makes it very appealing for robotic applications. However, there is still room for improvement. For instance, the strategy for learning POs does not ensure that the planner will be able to find the optimal plan for each situation it faces. This is because no further POs will be generated if the planner finds a plan that is sufficient to achieve a goal without unexpected effects. To ensure that the system can generate optimal plans, it will be necessary to include some random exploration of actions beyond those provided by the teacher, which we did not consider in this study. This could be implemented in future extensions of our iDMF.

Another point to consider is the strategy used for generating POs from the observed changes in the state transition. Although this is a common strategy that is employed by other approaches for learning POs [27, 22, 4], the assumption that all the observed changes are caused by the executed action limits the applicability of the framework to controlled environments where the frame problem is not significant. We consider that the proposed probabilistic approach prevents POs that code non-causative changes from being used for planning. This is because they would converge to very low probabilities as more experiences are gathered, and thus they would be replaced by more confident POs. However, this conjecture still remains to be proven.

We evaluated the performance of our framework using two examples. First, we used the Sokoban game, which is a well-known benchmark for evaluating planning and learning paradigms. This allowed us to control the complexity of the problem so we could perform a rigorous evaluation of the method. Our robotics-based implementation of this game might not be of immediate relevance, but many human-like tasks require the movement of objects in a specific order, similar to the Sokoban game. Thus, the transfer of our results to such scenarios is possible. However, the second application tested in this study, i.e., table rearrangement using ARMAR III, is of direct relevance to robotics because these types of pick-and-place actions occur generically and very often in service scenarios, as well as in industrial robotic scenarios.

7. Conclusion

In this study, we showed that AI techniques can be efficiently integrated into robotic platforms to learn and execute human-like tasks by exploiting natural teacher-learner interactions involving a human. Whenever a robotic execution plan halts, the human teaches an appropriate action and the robot can then complete the task by itself. For a given task, our results show that this type of online learning facilitates rapid convergence on a set of POs, which finally allows the agent to perform the task autonomously.

In this study, our main focus was only to instruct actions. This is easier and more typical of human-human cooperative work, rather than always instructing both actions and (all) action consequences. The robot was allowed to discover the cause-effect relationships of these actions autonomously to complete the missing POs. A major feature of the proposed method that allows correct POs to be found is our development of an efficient approach that permits the robot to determine unchanged causative conditions as well as the actions that lead to the observed changes. We showed that this problem can be solved using an easy-to-implement approach that evaluates different combinations of conditions in parallel while considering the lack of experience, which is fundamental for preventing biased estimates when learning based on a few trials.

In this manner, we suggest that simple natural human instructions can be used to trigger task-relevant actions, while plain fast learning can be used to immediately generate POs simply from observed state transitions to rapidly increase the autonomy of the robot.

Thus, the main contribution of this study is the development of a method that can help make robots part of our world, where they might help us rapidly without requiring tedious programming or complex interactions.

Acknowledgments

We thank Prof. Tamim Asfour, Prof. Rüdiger Dillmann, and their team at the Karlsruhe Institute of Technology for all the support provided with the implementation of the system on the ARMAR platform. This research has received funding from the European Communitys Seventh Framework Programme FP7/2007-2013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) under grant agreement no. 600578, ACAT.

References

[1] Agostini, A., Torras, C., Wörgötter, F., 2011. Integrating Task Planning and Interactive Learning for Robots to Work in Human Environments. In: Proceeding of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11). Barcelona, Spain. pp. 2386–2391.

[2] Argall, B. D., Chernova, S., Veloso, M., Browning, B., 2009. A survey of robot learning from demonstration. Robotics and autonomous systems 57 (5), 469–483.

[3] Asfour, T., Azad, P., Vahrenkamp, N., Regenstein, K., Bierbaum, A., Welke, K., Schroeder, J., Dillmann, R., 2008. Toward humanoid manipulation in human-centred environments. Robotics and Autonomous Systems 56 (1), 54 – 65.

[4] Benson, S., 1995. Inductive learning of reactive action models. In: Proc. of the Twelfth Int. Conf. on Machine Learning. Morgan Kaufmann, pp. 47–54.

[5] Billard, A., Calinon, S., Dillmann, R., Schaal, S., 2008. Robot programming by demonstration. In: Siciliano, B., Khatib, O. (Eds.), Springer Handbook of Robotics. Springer Berlin Heidelberg, pp. 1371–1394.

[6] Botea, A., Müller, M., Schaeffer, J., 2003. Using abstraction for planning in sokoban. Computers and Games, 360–375.

[7] Cestnik, B., 1990. Estimating probabilities: A crucial task in machine learning. In: Proc. of the Ninth European Conference on Artificial Intelligence. Vol. 1990. pp. 147–9.

[8] Clark, P., Boswell, R., 1991. Rule induction with CN2: Some recent improvements. In: Proc. of the Fifth European Working Session on Learning. pp. 151–163.

[9] Fern, A., Khardon, R., Tadepalli, P., 2011. The first learning track of the international planning competition. Machine Learning 84 (1-2), 81–107.

[10] Furnkranz, J., Flach, P., 2003. An analysis of rule evaluation metrics. In: Proc. of the Twentieth Int. Conf. on Machine Learning. Vol. 20. pp. 202–209.

[11] Ghallab, M., Nau, D., Traverso, P., 2004. Automated Planning Theory and Practice. Elsevier Science.

[12] Gil, Y., 1994. Learning by experimentation: incremental refinement of incomplete planning domains. In: Proc. of the Eleventh Int. Conf. on Machine Learning.

[13] John, G., Langley, P., 1995. Estimating continuous distributions in bayesian classifiers. In: Proceedings of the eleventh conference on uncertainty in artificial intelligence. pp. 338–345.

[14] Junghanns, A., Schaeffer, J., 1997. Sokoban: A challenging single-agent search problem. In: In IJCAI Workshop on Using Games as an Experimental Testbed for AI Reasearch. Citeseer.

[15] Kemp, C., Edsinger, A., Torres-Jara, E., 2007. Challenges for robot manipulation in human environments. IEEE Robotics and Automation Magazine 14 (1), 20.

[16] LaValle, S., 2006. Planning algorithms. Cambridge Univ Pr.

[17] Mitchell, T. M., 1997. Machine learning. 1997. Burr Ridge, IL: McGraw Hill 45.

[18] Nicolescu, M., Mataric, M., 2003. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In: Proceedings of the second international joint conference on Autonomous agents and multiagent systems. ACM, pp. 241–248.

[19] Petrick, R., Bacchus, F., 2002. A knowledge-based approach to planning with incomplete information and sensing. In: AIPS. pp. 212–221.

[20] Russell, S., Norvig, P., 2009. Artificial intelligence: A modern approach.

[21] Rybski, P., Yoon, K., Stolarz, J., Veloso, M., 2007. Interactive robot task training through dialog and demonstration. In: Human-Robot Interaction (HRI), 2007 2nd ACM/IEEE International Conference on. IEEE, pp. 49–56.

[22] Shen, W., 1989. Rule creation and rule learning through environmental exploration. In: Proc. of the Eleventh Int. Joint Conf. on Artificial Intelligence. Morgan Kaufmann, pp. 675–680.

[23] Szeliski, R., 2010. Computer vision: algorithms and applications. Springer.

[24] Thrun, S., Bala, J., Bloedorn, E., Bratko, I., Cestnik, B., Cheng, J., De Jong, K., Dzeroski, S., Fisher, D., Fahlman, S., et al., 1991. The MONK's problems: A Performance Comparison of Different Learning Algorithms (CMU-CS-91-197).

[25] Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E., Blythe, J., 1995. Integrating planning and learning: The prodigy architecture. Journal of Experimental & Theoretical Artificial Intelligence 7 (1), 81–120.

[26] Walsh, T., Littman, M., 2008. Efficient learning of action schemas and web-service descriptions. In: Proc. of the Twenty-Third AAAI Conf. on Artificial Intelligence. pp. 714–719.

[27] Wang, X., 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. In: ICML. pp. 549–557.
[28] Wang, X., 1996. Planning while learning operators. In: AIPS. pp. 229–236.