Planning for Learning Object Properties *

Leonardo Lamanna[1,2], Luciano Serafini[2], Mohamadreza Faridghasemnia[3], Alessandro Saffiotti[3], Alessandro Saetti[2], Alfonso Gerevini[2], Paolo Traverso[1]

[1] Fondazione Bruno Kessler, Trento, Italy
[2] Department of Information Engineering, University of Brescia, Italy
[3] Center for Applied Autonomous Sensor Systems, University of Örebro, Sweden
llamanna@fbk.eu, serafini@fbk.eu, mohamadreza.farid@oru.se, asaffio@aass.oru.se, alessandro.saetti@unibs.it, alfonso.gerevini@unibs.it, traverso@fbk.eu

Abstract

Autonomous agents embedded in a physical environment need the ability to recognize objects and their properties from sensory data. Such a perceptual ability is often implemented by supervised machine learning models, which are pre-trained using a set of labelled data. In real-world, open-ended deployments, however, it is unrealistic to assume to have a pre-trained model for all possible environments. Therefore, agents need to dynamically learn/adapt/extend their perceptual abilities online, in an autonomous way, by exploring and interacting with the environment where they operate. This paper describes a way to do so, by exploiting symbolic planning. Specifically, we formalize the problem of automatically training a neural network to recognize object properties as a symbolic planning problem (using PDDL). We use planning techniques to produce a strategy for automating the training dataset creation and the learning process. Finally, we provide an experimental evaluation in both a simulated and a real environment, which shows that the proposed approach is able to successfully learn how to recognize new object properties.

Introduction

Agents embedded in a physical environment, like autonomous robots, need the ability to perceive objects in the environment and recognize their properties. For instance, a robot operating in an indoor environment should be able to recognize whether a certain box found in the environment is open or closed. From these perceptions the agent can build and use abstract representations of the states of the environment to reach its goals by automatic planning techniques. The common approach to provide an agent with such perceptual capabilities consists in pre-training offline a (set of) perception models from hundreds of thousands of annotated data (e.g., images or other sensory data). See for instance (Asai 2019; Dengler et al. 2021; Lamanna et al. 2022).

In offline training approaches, the perception capabilities are fixed once and for all. This is in stark contrast with a main requirement in many robotics applications: agents embedded in real-world, open-ended environments should be able to dynamically and autonomously improve their perceptual abilities by actively exploring their environments. This is also in agreement with the emerging popular research area of interactive perception (Bohg et al. 2017). When perception functions are modelled by (deep) neural networks, an open and interesting challenge is whether agents can autonomously decide when and how to improve their perception models, by collecting the needed training data and using them to train the neural network.

In this paper, we explore a way to address this challenge with an automated planning approach. In particular, we design a PDDL planning domain (McDermott et al. 1998) for *planning to learn (or improve) the perceptual capabilities of the agent*. We focus on the problem of automatically training neural networks able to recognize properties of objects, e.g., open/closed, by relying on a pre-trained object detector. We extend a PDDL planning domain, called *base domain*, with new actions and predicates for learning properties of object types. Such an extension, called *learning domain*, is specified in a meta-language of the base language. It contains the reification of properties and types of the base language. For instance, if `Is_Open` is a property of the base domain, the learning domain contains the object 'is_open' of type `Property`. Furthermore, the learning domain contains actions for collecting training examples for properties, and actions for training the network with them.

The online learning of object properties is obtained by planning in the union of the base and learning domains, and by executing the generated plan. The base domain allows the agent to plan to reach a state where the agent can observe an object with a certain property, e.g., a state where $\texttt{Is\_Open}(\texttt{box}_0)$ is true. The learning domain allows the agent to plan for actions that collect observations of objects with the property being true, e.g., take pictures of $\texttt{box}_0$, which is known to be open, and add them to the positive training examples for the property `Is_Open`. In this way, the agent automatically maps low level perceptions (e.g., images of an open box) into the symbolic property of objects at the abstract planning level (e.g., "the box is open" in PDDL).

We provide an experimental evaluation both in a photo-realistic simulated environment (Kolve et al. 2017) and in a real world setting. In the simulated environment, we evaluate the ability to learn certain object properties under two different conditions of object detection: noisy and ground truth. The results indicate that the approach is able to successfully

*This paper has been already published in the proceedings of AAAI 2023.

learn and recognize the object properties with high precision/recall for most object classes. In the real world environment, actions for changing object properties (e.g., opening a book) are performed cooperatively with humans, e.g., by asking a human to physically perform the required action.

Related Work

As far as we know, the proposed approach is new. Several approaches are based on the idea of exploiting automated learning for different planning tasks, like for learning action models, heuristics, plans, or policies. Most of the research on learning action models, see, e.g., (Aineto, Jiménez, and Onaindia 2018, 2019; Bonet and Geffner 2020; McCluskey et al. 2009; Cresswell, McCluskey, and West 2013; Gregory and Cresswell 2015; Lamanna et al. 2021; Juba, Le, and Stern 2021), does not deal with the problem of learning from real value perceptions. Other works have addressed the problem of learning planning domains from perceptions in the form of high dimensional raw data (such as images), see, e.g., (Asai and Fukunaga 2018; Asai 2019; Janner et al. 2018; Dengler et al. 2021; Konidaris, Kaelbling, and Lozano-Pérez 2018; Liberman, Bonet, and Geffner 2022). In these works, the abstract planning domain is obtained by offline pre-training, and the mapping between perceptions to the abstract model is fixed, while we learn/adapt this mapping online. Our approach shares some similarities with the work on planning by reinforcement learning (RL) (Sutton and Barto 2018), since we learn by acting in a (simulated) environment, and especially with the work on deep RL (DRL) (Mnih et al. 2015, 2016), which dynamically trains a deep neural network by interacting with the environment. However, DRL focuses on learning policies and perceptions are mapped into state embeddings that cannot be easily mapped into human comprehensible symbolic (PDDL) states. In general, while all the aforementioned works address the problem of using learning techniques for planning, we address a different problem, i.e., using planning techniques for learning automatically and online to recognize object properties from low level high dimensional data.

We share a similar motivation of the research on interactive perception, see (Bohg et al. 2017) for a comprehensive survey, and especially the work in this area for learning properties of objects, see, e.g., (Natale, Metta, and Sandini 2004). However, most of this work has the objective to integrate acting and learning, and to study the relation between action and sensory response. We instead address the challenge of building autonomous systems with planning capabilities that can automate the training and learning process.

A wide variety of approaches and applications have been proposed for enhancing robotic agents with active learning techniques (Kulick et al. 2013; Cakmak and Thomaz 2012; Cakmak, Chao, and Thomaz 2010; Chao, Cakmak, and Thomaz 2010; Ribes et al. 2015; Hayes and Scassellati 2014; Huang, Jin, and Zhou 2010; Ashari and Ghasemzadeh 2019). In these works, the robotic agents improve their skills or learn new concepts by collecting and labeling data in an online way. However, all these methods label data by means of either human supervisions or a confidence criteria applied on the prediction of a pre-trained model. In contrast, our approach does not require human supervision, and collected data are labelled by applying actions.

In (Ugur and Piater 2015), similarly to our approach, the authors propose a method for learning the predicates corresponding to action effects after their executions. They learn action effects by clustering hand-crafted visual features of the manipulated objects, extracted from the continuous observations obtained after executing the actions. However, the learned predicates lack of interpretation, which must be given by a human. On the contrary, our approach learns explainable predicates. Moreover, they focus on learning the effects of a single action (i.e., stacking two blocks) in a fully observable environment using ground truth object detection. Our approach learns predicates corresponding to the effects of several actions in partially observable environments, and using a noisy object detector.

The approach by (Migimatsu and Bohg 2022) learns to map images into the truth values of predicates of planning states. Differently from us, their approach is offline and requires the sequence of images labeled with actions, while our approach plans for generating this sequence online. We share the idea of learning state representations through interaction with (Pinto et al. 2016), where they learn visual representations of an environment by manipulating objects on a table. Notably, they learn the visual representation in an unsupervised way, through a CNN trained on a dataset generated by interacting with objects. However, the learned representations lack of interpretation. Furthermore, in (Pinto et al. 2016), the learned representations are not suitable for applying symbolic planning.

Finally, our extension of the planning domain with a meta-language shares some commonalities with the work on planning at the knowledge level (see, e.g., (Petrick and Bacchus 2002)), which addresses the different problem of planning with incomplete information and sensing.

Preliminaries

Symbolic planning A planning domain $\mathcal{D}$ is a tuple $\langle \mathcal{P}, \mathcal{O}, \mathcal{H} \rangle$ where $\mathcal{P}$ is a set of first order predicates with associated arity, $\mathcal{O}$ is a set of operators with associated arity, and $\mathcal{H}$ associates to every operator $op \in \mathcal{O}$ an action schema. The action schema is composed of a triple $\langle \mathsf{par}(op), \mathsf{pre}(op), \mathsf{eff}^+(op), \mathsf{eff}^-(op) \rangle$ in which $\mathsf{par}(op)$ is a n-tuple of distinct variables $x_1, \ldots, x_n$, where n is the arity of op, and $\mathsf{pre}(op)$ is a set of first order formulas with predicates in $\mathcal{P}$ and arguments in $x_1, \ldots, x_n$, and $\mathsf{eff}^+(op)$ and $\mathsf{eff}^-(op)$ are set of *atomic* first order formula with predicates in $\mathcal{P}$ and arguments in $x_1, \ldots, x_n$. Among the unary predicates of $\mathcal{P}$, we distinguish between object types and object properties, which are denoted with t and p, respectively.

Given a planning domain $\mathcal{D}$ and a set of constants $\mathcal{C}$, a grounded planning domain is obtained by grounding all the actions schema of $\mathcal{D}$ with the constants in $\mathcal{C}$. The set of states of a grounded planning domain $\mathcal{D}(\mathcal{C})$ is the set of all possible subsets of atoms that can be built by instantiating every predicate in $\mathcal{P}$ in all possible ways with the constants in $\mathcal{C}$. A ground action model defines a transition func-

tion among states where $(s, op(c_1, \ldots, c_n), s')$ if and only if $s \models \mathsf{pre}(op(c_1, \ldots, c_n))$ and $s' = s \cup \mathsf{eff}^+(op(c_1, \ldots, c_n)) \setminus \mathsf{eff}^-(op(c_1, \ldots, c_n))$.

A planning problem Π on a grounded planning domain $\mathcal{D}(\mathcal{C})$ is a triple $\Pi = \langle \mathcal{D}(\mathcal{C}), s_0, g \rangle$, where s_0 is an initial state and g is a first order formula that identifies a set of states of $D(\mathcal{C})$ in which the goal is satisfied, i.e., $\{s \in 2^{\mathcal{P}(\mathcal{C})} \mid s \models g\}$. A plan π for problem Π is a sequence of ground actions $\langle a_1, \ldots, a_k \rangle$ such that (s_{i-1}, a_i, s_i) for $i = 1, \ldots, k$ is a transition of $\mathcal{D}(\mathcal{C})$ and $s_k \models g$.

Perception functions The agent perceives the environment by sensors that return real-value measurements on some portion of the environment. For example, the perception of an agent with a on-board camera and a system for estimating its position consists in a vector (x, y, z) of coordinates and an RGB-D image taken by the agent's on-board camera. Observations are partial (e.g., the camera provides only the front view) and could be incorrect (e.g., the estimation of the position could be noisy). We suppose that, at the time when a perception occurs, the agent's knowledge about the environment is represented by a grounded planning domain $\mathcal{D}(\mathcal{C})$, where C represents the set of objects already discovered in the environment. Each $c \in \mathcal{C}$ is associated with an anchor (Coradeschi and Saffiotti 2003) that describes the perceptual features of c that have been collected by the agent so far (e.g., the pictures of c from different angles, the estimated position and size of c, etc.). At the beginning, the set $\mathcal{C}$ of constants is empty. The agent performs and processes each perception in order to extract some knowledge about the objects in the environment, and about their properties in the current state. This is achieved by combining an *object detector* and a set of *property classifiers*.

The object detector identifies a set of objects in the current perception (e.g., RGB-D image) and predicts their types (i.e., it selects one type among the object types of the planning domain). Every detected object is associated with numeric features (e.g., the bounding box, the estimation of the position, etc.), which are used to build the anchors of the detected object. The features of each detected object are compared with the features of the objects already known by the agent, i.e., those present in the current set of constants $\mathcal{C}$. If the features of the detected object matches (to a certain degree) the features of a $c \in \mathcal{C}$, then the features of c are updated with the new discovered features. Otherwise $\mathcal{C}$ is extended with a new constant c anchored to the features of the detected object, and the type $t(c)$ is asserted in the planning domain, where t is the type returned by the object detector.

For every object c of type t returned by the object detector and for every property p that applies to t, a classifier $\rho_{t,p}$ predicts if c has/has not the property p. Notice that not all properties apply to a type, e.g., a laptop cannot be filled or empty. Furthermore, for the same property we use different classifiers for different types, since predicting that a bottle is open or that a book is open from visual features are two very different tasks. $\rho_{t,p}$ can be specified either explicitly by a set of predefined rules, or it can be a machine learning model trainable by supervised examples. For instance, the classifier that checks if an object is `Close_To` the agent is defined by a threshold on the distance between the agent and the object position. Other properties (e.g., `Is_Open`) are predicted using a neural network, which takes as input object images and returns the probability of the property being true.

Plan execution To achieve its goal (expressed in a formula of the language of the planning domain), the agent generates a plan using a classical planner (e.g., we used Fast-Forward (Hoffmann 2001)), and then it executes the plan. However, the symbolic actions of the plan need to be translated into sequences of *operations* executable by the agent's actuators (e.g., rotate of 30°, grasp the object in position x, y, z, move forward of 30cm). Designing effective and robust methods for producing this mapping is a research area which goes out of the scope of this paper, see for instance (Eppe, Nguyen, and Wermter 2019). In our experiments, we adopt state-of-the-art path planning algorithms (based on a map learned online by the agent) and ad-hoc compilations of actions. However, it is worth noting that we do not assume the execution of the actions leads to the symbolic state predicted by the planning domain. For instance, the execution of the action `Go_Close_To`(c) might end up in a situation where the agent is not close enough to the object c and the predicate `Close_To`(c) is false, despite being a positive effect of the action `Go_Close_To`(c). Moreover, the execution of a symbolic action can have effects that are not predicted by the action schema. For instance some properties of an object might become true even if they are not in the positive effects of the symbolic actions. For these reasons, after action executions, the agent must check if the plan is still valid, and if not, it should react to the unexpected situation, e.g., by replanning.

Problem Definition

We place an agent in a random position of an unknown environment; we initialize it with the following components: (i) a set of sensors on the environment; (ii) a trained object detector ρ_o; (iii) a planning domain $\mathcal{D} = (\mathcal{P}, \mathcal{O}, \mathcal{H})$; (iv) a method for executing its ground actions; (v) *an untrained neural network* $\rho_{t,p}$ for predicting the property p of the objects of type t, for a subset of pairs (t, p) of interest.

We focus on the online training of the $\rho_{t,p}$'s; our aim is to design a general method to autonomously generate symbolic plans for producing a training set $T_{t,p}$, for every pair (t, p) of interest, and use $T_{t,p}$ to train the perception function $\rho_{t,p}$.

$T_{t,p}$ contains pairs (c, v), where c is (the name of) an object of type t with the associated anchor (e.g., the visual features of the object) and $v \in \{p, \neg p\}$ is the value of the property p. Since $T_{t,p}$ is automatically created by acting in the environment, it may contain wrong labels. We evaluate the effectiveness of our method on the performance (precision and recall) of each $\rho_{t,p}$ against a ground truth data set collected independently by the agent.

Method

We explain the proposed method with a simple example. Suppose an agent aims to learn to recognize the property `Is_Turned_On` for objects of type `Tv`, it can proceed as follows: *(i)* look for an object (say $\texttt{tv}_0$) of type `Tv`; *(ii)* turn $\texttt{tv}_0$ on to make sure that `Is_Turned_On`($\texttt{tv}_0$) is true, *(iii)* take

Observe(o, t, p):	
pre:	$\neg$Viewed(o, t, p)
	Closed_To(o)
	Known(o, t, p)
eff$^+$:	Sufficient_Obs(t, p)
	Viewed(o, t, p)
Explore_for(t, p)	
pre:	$\forall x$(Known$(x, t, p) \rightarrow$ Viewed(x, t, p))
	$\neg$ Sufficient_Obs(t, p)
eff$^+$:	Explored_for(t)
Train(t, p, q):	
pre	Sufficient_Obs(t, p)
	Sufficient_Obs(t, q)
eff$^+$:	Learned(t, p, q)

Table 1: Schemas for Observe, Explore_for and Train.

pictures of $\texttt{tv}_0$ from several perspectives, and label them as positive examples for Is_Turned_On. To produce negative examples for the same property, the agent can proceed in the same fashion, applying the action Turn_Off($\texttt{tv}_0$).

The behaviour explained above should be automatically produced and executed by the agent for every learnable pair (t, p), where t denotes an object type and p a learnable property. Therefore, in the following, we explain a procedure that extends automatically the planning domain of the agent to express the goal of learning p for t, and such that the procedure for collecting training data for $\rho_{t,p}$ is generated by a symbolic planner, and can be executed by the agent. This method requires that, for every learnable pair (t, p), the planning domain contains at least an operator applicable to object of type t that makes p true, and one that makes p false.

This means that we have to extend the planning domain with the capability of expressing facts about its properties and types, i.e., we have to extend it with meta predicates and names for the elements of the planning domain $\mathcal{D}$.

Extended Planning Domain for Learning

Table 1 summarizes how we extend the planning domain for observing, exploring and learning.

Names for types and properties For each object type $t \in \mathcal{P}$ (e.g., Box), we add a new constant 't' (e.g., 'box')[1]. For each object property $p \in \mathcal{P}$ (e.g., Is_Open), we add two new constants, namely 'p' and 'not_p' (e.g., 'is_open' and 'not_is_open').

Epistemic predicates We extend $\mathcal{P}$ with predicates for stating that an agent knows/believes that an object has a certain property in a given state. The binary predicate Known(o,'p') (resp. Known(o, 'not_p')) indicates that the agent knows that the object o has (resp. does not have) the property p. The atom Known(x,'p') is automatically added to the positive (resp. negative) effects of all the actions that have $p(x)$ in their positive (resp. negative) effects; similarly, the atom Known(x,'not_p') is automatically added to the positive (resp. negative) effects of all the actions that have $p(x)$ in their negative (resp. positive) effects. For example, the atoms Known(x, 'is_turned_on') and Known(x, 'not_is_turned_on') are added to the positive and negative effects of Turn_On(x), respectively. Similarly, the atoms Known(x, 'is_turned_on') and Known(x, 'not_is_turned_on') are respectively added to the negative and positive effects of Turn_Off(x).

Predicates and operators for observations We extend the planning domain with the operator Observe(o, t, p), which takes as input an object o, a type t, and a property p. The low level execution of Observe(o, t, p) consists in extending the training dataset $T_{t,p}$ with observations (i.e., images) of object o taken from different perspectives. The positive effects of Observe(o, t, p) contain the atom Viewed(o, p), and the preconditions of Observe(o,t,p) contain the atom $\neg$Viewed(o, p), which prevents the agent from again observing o for the property p in the future.

The atom Sufficient_Obs(t, p) is added to the positive effects of the action Observe(o, t, p). Whether the agent, after executing Observe(o, t, p), has not collected enough observations of objects of type t with property p, the atom Sufficient_Obs(t, p) is actually false, in contrast with what is predicted by the planning domain, and the agent has to plan for observing other objects of type t.

Predicates and operators for exploration The planning domain is extended with the binary operator Explore_for(t, p) that explores the environment looking for new objects of type t. The precondition of Explore_for(t, p) is that all the known objects of type t have been viewed for the property p, i.e., $\forall x$(Knows$(x, t, p) \rightarrow$ Viewed(x, t, p)). Indeed, finding a new object creates a new object o in the planning domain, and makes aware the agent that properties of o can be observed. Explored_for(t) is a positive effect of Explore_for(t, p) in the planning domain. Such an effect indicates that the environment has been (even partially) explored for finding new objects of type t. However, the actual execution of Explore_for(t, p) will not make it true until the environment has been completely explored, or a maximum number of iterations has been reached.

Predicates and operators for learning We extend the planning domain with the predicate Learned$(t, p,$ not_$p)$, indicating if the agent has collected enough observations, and trained $\rho_{t,p}$. We add to the planning domain the operator Train(t, p, q). When the agent executes the action Train(t, p, q), the network $\rho_{t,p}$ is trained using $T_{t,p}$ as positive examples and $T_{t,q}$ as negative examples. The preconditions of this action include Sufficient_Obs(t, p) and Sufficient_Obs(t, q) that guarantee to have sufficient positive and negative examples for training $\rho_{t,p}$. This action has only one positive effect, which is Learned(t, p, q).

Specifying the goal formula In the extended planning domain, the goal formula g for learning a property p for an object type t is defined as:

$$g = \texttt{Learned}(t, p, \texttt{not\_}p) \vee \texttt{Explored\_for}(t). \quad (1)$$

For example, suppose that an agent aims to learn the property Turned_On for objects of type Tv, then

[1] Quotes are used to indicate names for elements of $\mathcal{P}$.

Algorithm 1: PLAN AND ACT TO LEARN OBJECT PROPS

Input: $\mathcal{D} = (\mathcal{P}, \mathcal{O}, \mathcal{H})$ a planning domain
Input: $g = \bigwedge_{(t,p)\in TP}(\texttt{Learned}(t,p) \vee \texttt{Explored\_For}(t))$

1: extend $\mathcal{D}$ with actions and predicates for learning
2: $\mathcal{C} \leftarrow$ names for types and properties in $\mathcal{P}$
3: $s \leftarrow \emptyset$
4: $T_{TP} \leftarrow \{T_{t,p} = \emptyset \mid (t,p) \in TP\}$
5: $\rho_{TP} \leftarrow \{\rho_{t,p} = \text{random init.} \mid (t,p) \in TP\}$
6: $\pi \leftarrow \text{PLAN}(\mathcal{D}(\mathcal{C}), s, g)$
7: **while** $\pi \neq \langle\rangle$ **do**
8: $\quad op \leftarrow \text{POP}(\pi)$
9: $\quad s \leftarrow s \cup \mathsf{eff}^+(op) \setminus \mathsf{eff}^-(op)$
10: $\quad \mathcal{C}, T_{TP}, \rho_{TP} \leftarrow \text{EXECUTE}(op)$
11: $\quad s \leftarrow \text{OBSERVE}()$
12: $\quad \pi \leftarrow \text{PLAN}(\mathcal{D}(\mathcal{C}), s, g)$
13: **end while**

$g =$ `Learned('tv', 'turned_on', 'not_turned_on')` $\vee$ `Explored_for('tv')`. If the current set of constants contains an object, say $\texttt{tv}_0$, of type `Tv` such that `Viewed(`$\texttt{tv}_0$`,'tv','is_turned_on')` and `Viewed(`$\texttt{tv}_0$`,'tv','not_is_turned_on')` are both false, then the goal is reachable by the plan:

```
Go_Close_To(tv0)
Turn_On(tv0)
Observe(tv0, 'tv','turned_on')
Turn_Off(tv0)
Observe(tv0,'tv', 'not_turned_on')
Train('tv', 'turned_on','not_turned_on').
```

After the execution of all the actions but the last one of the above plan, if the agent has not collected enough training data for 'turned_on' and 'not_turned_on', the atoms `Sufficient_Obs('tv','turned_on')` and `Sufficient_Obs('tv','not_turned_on')` will be false, and the last action of the plan cannot be executed. In such a case, the agent has to replan in order to find another tv which has not been observed yet.

Finally, notice that whether all the TVs known by the agent have been observed for the property `Turned_On`, then the formula $\forall x(\texttt{Known}(x,\text{'tv'},\text{'turned\_on'}) \rightarrow \texttt{Viewed}(x,\text{'tv'},\text{'turned\_on'})$ is true, and the goal can be achieved by generating a plan that satisfies `Explored_for('tv')`, i.e., by executing the action `Explore_For('tv','turned_on')`, which explores the environment for new TVs.

Main control cycle

The main control cycle of the agent is described in Algorithm 1, which takes as input a planning domain $\mathcal{D}$ and the goal g for learning a set TP of type-property pairs. At the beginning, the set of constants $\mathcal{C}$ contains only the names for types and properties, and the state s is empty (lines 2–3). For every pair $(t,p) \in TP$, the algorithm initializes the training set $T_{t,p}$ to the empty set, and the neural networks $\rho_{t,p}$ (lines 4–5). Then, a plan π is generated (line 6). In the while loop (lines 7–13), the state s is updated according to the action schema (line 9). Next, the first action of the plan is executed and the set of known constants $\mathcal{C}$, the datasets $T_{t,p}$, and the neural networks $\rho_{t,p}$ are updated (line 10). Notice that, since the perceived effects of actions might not be consistent with those contained in the action schema, a sensing using the not trainable perception functions is necessary, and the state is updated accordingly (line 11). Moreover, since π might be no more valid in the updated state, a new plan must be generated (line 12). The algorithm terminates if either the whole environment has been explored or a maximum number of iterations has been reached, since, in such cases, the atom `Explored_for`(t) is set to true, and plan π for g is empty.

Experimental Evaluation

We evaluate our approach on the task of collecting a dataset and training a set of neural networks to predict the four properties `Is_Open`, `Dirty`, `Toggled`, and `Filled` on 32 object types, resulting in 38 pairs (t,p), since not all properties are applicable to all object types.

Simulated environment We experiment our approach in the ITHOR (Kolve et al. 2017) photo-realistic simulator of four types of indoor environments: kitchens, living-rooms, bedrooms, and bathrooms. ITHOR simulates a robotic agent that navigates the environment and interacts with the objects by changing their properties (e.g., opening a box, turning on a tv). The agent has two sensors: a position sensor and an on-board RGB-D camera. For our experiment we split the 120 different environments, provided by ITHOR, into 80 for training, 20 for validation, and 20 for testing. Testing environments are evenly distributed among the 4 room types.

Object detector For the object detector ρ_o, we used the YoloV5 model (Jocher et al. 2021), which takes as input an RGB image and returns the object types and bounding boxes detected in the input image. For training ρ_o, we have generated the training (and validation) sets by randomly navigating in the training (and validation) environments, and using the ground truth object types and bounding boxes provided by ITHOR. The training and validation sets contain 115 object types and are composed by 259859 and 56190 examples, respectively. For validating the object detector, we performed 300 runs (with 10 epochs for each run) of the genetic algorithm proposed in (Jocher et al. 2021).

Property predictors For the perception functions $\rho_{t,p}$ predicting properties, we adopted a ResNet-18 model (He et al. 2016) with an additional fully connected linear layer, which takes as input the RGB image of the object and returns the probability of p being true for the object. We consider that the input object has the property p if the probability is higher than a given threshold (set to 0.5 in our experiments).

Evaluation metrics and ground truth We evaluate each trained $\rho_{t,p}$ using precision and recall against a test set $G_{t,p}$, obtained by randomly navigating the 20 testing environments and using the ground truth information provided by ITHOR. In particular, for the `Is_Open` property we generated a test set with 8751 examples, 2512 for the `Toggled` property, 1310 for the `Filled` property, and 3304 for the `Dirty` one. It is worth noting that the size of the test set

	size of $G_{t,p}$		size of $T_{t,p}$		Precision		Recall	
Object type	ND	GTD	ND	GTD	ND	GTD	ND	GTD
			Dirty					
bed	564	564	1502	671	0.95	0.57	0.43	0.61
bowl	280	280	383	1027	0.67	0.98	0.81	0.73
cloth	96	210	61	503	0.93	0.95	0.78	0.7
cup	96	262	146	986	0.63	0.99	0.95	0.54
mirror	654	678	2490	3100	0.91	0.9	0.68	0.8
mug	230	432	225	1367	0.88	0.94	0.42	0.74
pan	140	200	20	476	0.76	0.99	0.87	0.79
plate	166	406	47	1304	0.61	0.97	0.97	0.77
pot	210	272	51	929	0.76	0.99	0.91	0.98
Weighted avg	-	-	-	-	0.84	0.89	0.68	0.74
			Filled					
bottle	22	22	78	150	0.65	0	1	0
bowl	328	256	390	1091	0.64	1	0.73	0.77
cup	116	286	200	1028	0.92	0.9	0.56	0.68
houseplant	34	34	18	72	0.5	0.5	0.65	0.82
kettle	-	84	-	337	-	0.25	-	0.4
mug	126	354	250	1136	0.8	0.86	0.51	0.56
pot	226	274	93	809	0.67	1	0.89	0.79
Weighted avg	-	-	-	-	0.7	0.86	0.72	0.66
			Is_Open					
book	148	268	367	1471	1	0.94	0.76	0.81
box	204	204	959	1044	0.92	0.88	0.37	0.54
cabinet	2892	2892	1545	1669	0.81	0.8	0.74	0.79
drawer	3343	3747	1237	2624	0.79	0.75	0.77	0.71
fridge	400	400	803	1109	0.78	0.81	0.72	0.75
laptop	360	360	1124	1531	0.93	0.97	0.85	0.82
microwave	250	250	742	843	0.68	0.82	0.5	0.68
showercurtain	144	134	271	567	0.47	0.96	0.41	0.76
showerdoor	74	140	56	346	0.88	0.71	0.19	0.98
toilet	356	356	1024	1148	0.89	0.9	0.63	0.74
Weighted avg	-	-	-	-	0.81	0.8	0.72	0.75
			Toggled					
candle	54	124	3	118	0.59	0.33	0.63	0.6
cellphone	-	216	-	682	-	0.84	-	0.94
coffeemachine	320	320	999	996	0.95	0.97	0.72	0.61
desklamp	12	56	254	255	1	0.91	1	0.97
desktop	-	56	-	184	-	1	-	0.93
faucet	602	480	921	1663	0.84	0.85	0.89	0.92
floorlamp	44	12	88	68	0.83	0.75	0.5	1
laptop	432	432	1545	1777	0.91	0.83	0.61	0.74
microwave	252	252	1131	1124	1	1	0.76	0.72
showerhead	-	46	-	12	-	1	-	1
television	222	238	269	510	0.99	0.94	0.85	0.95
toaster	280	280	713	1072	0.86	0.98	0.59	0.7
Weighted avg	-	-	-	-	0.9	0.88	0.74	0.8

Table 2: Size of the ground truth test set $G_{t,p}$, the generated training set $T_{t,p}$, and performance in terms of precision and recall on the 38 type-property pairs.

for the `Is_Open` property is higher than other ones since the number of object types that can be open is higher than the ones with other properties.

Experiments with the Simulator

We run our approach in each testing environment for training the neural network model associated to $\rho_{t,p}$ with the training set $T_{t,p}$ collected online. At each run, the agent starts in a random position of the environment and executes 2000 iterations, where at each iteration a low-level operation (e.g., move forward of $30cm$) is executed.

To understand how the errors of the objects detector affect the performance, we propose two variants of our approach, namely ND (Noisy Detections) and GTD (Ground Truth Detections). In both variants, the agent trains $\rho_{t,p}$ on the training set $T_{t,p}$ collected in a single environment, and is evaluated on the test set $G_{t,p}$ previously generated in the same environment. In the ND variant, the agent is provided with a pre-trained object detector ρ_o; while in the GTD variant

Figure 1: Pepper taking images of a laptop and asking a human to manipulate it for learning the property `Folded`.

the agent is provided with a perfect ρ_o, i.e., the ground truth object detections provided by ITHOR. In both variants, the neural networks $\rho_{t,p}$'s are trained for 10 epochs with $1e^{-4}$ learning rate; the other hyperparameters are set to the default values provided by PyTorch1.9 (Paszke et al. 2019).

Experimental results We compare the versions ND and GTD for each learned property; the results are shown in Table 2. In particular, the columns of Table 2 contain the object type, the number of examples collected in the training and test sets, respectively $G_{t,p}$ and $T_{t,p}$, the metrics precision and recall averaged over all 20 environments. It is worth noting that the size of the test set can vary among ND and GTD, since we remove from the test set the object types that are missing in the training set, i.e., the object types that have not been observed by the agent. This is because we are interested in evaluating the learning performance on the object types that the agent actually manipulates and observes. Moreover, there are particular object types (e.g., desktop and showerhead in Table 2) that are never recognized by the object detector, hence they are missing in the training set, and they are assigned the '-' value in Table 2.

Table 2 shows the results obtained for learning properties `Dirty`, `Filled`, `Is_Open`, and `Toggled`. Not surprisingly, both the weighted average precision and recall of the GTD version are almost always higher than the ND ones, i.e., the overall learning performance are better when the agent is provided with ground truth object detections. The recall is generally lower than the precision, this is because for almost all object types, the number of negative examples is higher than the positive one, i.e., the training datasets are not balanced. Therefore, the agent is more likely to predict that a property is false, which causes more false negatives and a decrease of the recall. In our experiments, we tried to balance the observations in the collected training sets by randomly removing positive or negative examples, but we obtained worse performance. More sophisticated strategies might measure the information of each observation and remove the less informative ones; however, this problem is out of the scope of this paper.

In the `Dirty` property results, for all object types but bed, the number of examples in the training set is higher for GTD, as expected. The examples of beds in GTD are lower be-

cause in all bedrooms the agent focused on observing other object types. Indeed, for all other object types contained in bedrooms (i.e., cloth, mug and mirror), the examples collected by GTD are more than the ND ones.

Moreover, for the `Dirty` property, the precision obtained by GTD is significantly higher than the ND one, for almost all object types (i.e., 7 out of 9). For the mirror object type, the precision achieved by both ND and GTD is almost equal. Remarkably, for the bed object type, the precision of the GTD version is much lower than the ND one. This is because, for large objects such as beds, the GTD version is more likely to collect examples not representative for the properties to be learned. For instance, the agent provided with ground truth object detections recognizes the bed even when it sees just a corner of the bed, whose image is not significant for predicting whether the bed is dirty or not. Moreover, the examples of objects of type bed in the training set collected by ND is much higher than the GTD one.

The recall of the GTD version is not always higher than the ND one. In our experiments, we noticed that, for both ND and GTD, an high precision typically entails a low recall, and viceversa. This is because typically the agent collects more positive or negative examples of a single object type. For instance, the precision achieved by ND on object types bed, cloth, mirror and mug is high and the recall is low. Similarly, the recall achieved by ND on object types bowl, cup, pan, plate and pot is high and the precision is low. The recall obtained by GTD is lower than the precision for all object types but bed, where there is no significant difference. Overall, the weighted average metric values show good performance, i.e., our approach is effective for learning to recognize properties without any dataset given a priori as input. Similar considerations given for the `Dirty` property apply to results obtained for properties `Is_Open`, `Toggled` and `Filled`, reported in Table 2. However, it is worth noting that for the `Filled` property, the metric values obtained by both ND and GTD versions are particularly low for the object types houseplant and kettle. This is because, for the mentioned object types, the `Filled` property is hard to recognize from the object images. For instance, the fact that an object of type kettle is filled with water cannot be recognized from its image, since the water in the kettle is not visible from an external view such as the agent one. Furthermore, GTD with the object type bottle achieves 0 value of both precision and recall, this is a particular situation where the neural network associated to the `Filled` property never predicts false positives when evaluated on examples of objects of type bottle, hence precision and recall equals 0.

Finally, we compared the performance of the property predictors learned by ND (Table 2) with a baseline where property predictors are trained on data manually collected as $G_{t,p}$. The baseline achieves an overall precision and recall of 0.81 and 0.84, respectively; while our approach achieves an overall precision and recall of 0.81 and 0.71, respectively. These results show that the online learning can achieve a precision comparable to the offline setting where data are manually collected, while its recall gets worse as the prediction gives false negative results more often.

Object type	Property	Precision	Recall
bowl	`Empty`	0.63	0.98
laptop	`Folded`	0.97	1.00
book	`Is_Open`	1.00	0.99
cup	`Filled`	0.93	0.83
Weighted avg	-	0.88	0.95

Table 3: Precision and recall obtained by the neural networks predicting object properties in a real environment.

Real World Demonstrator

To test our method in a real-world setting, we used a Softbank's Pepper humanoid robot in PEIS home ecology (Saffiotti et al. 2008), shown in Figure 1. As an object detector, we adopted a publicly available model of YoloV5 pretrained on the MS-COCO dataset (Lin et al. 2014). For manipulation actions, Pepper asks a human to do the manipulations, due to its limited capabilities of manipulating objects. We used Pepper's speech-to-text engine for simple verbal interaction with the human. Given an object type and a property, Pepper first looks for the object and then asks the human about the property's state. Next, it collects samples and asks the human to change the state of the property, and after human confirmation, it further collects samples.

We run experiments for learning pairs (type, property) reported in Table 3. For each pair, we run the experiment 7 times with different objects of the same type. At each run, Pepper collects 100 examples of the observed property, grouped into 50 positive and 50 negative examples. For each property, we took 4 runs for training (i.e., 400 examples), and 3 runs for testing (i.e., 300 examples). Table 3 shows the precision and recall obtained on the test sets. Both the average precision and recall are high. For the simpler properties (i.e., `Is_Open` and `Filled`), Pepper almost perfectly learned to recognize them. These results demonstrate that our approach can be effective also in real world environments.

Conclusions and Future Work

We address the challenge of using symbolic planning to automate the learning of perception capabilities. We focus on learning object properties, assuming that an object detector is pretrained. We experimentally show that our approach is feasible and effective. Still a lot of work must be done to address the general problem of planning and acting to learn in a physical environment. For example, planning for online training the object detector or learning relations among different objects. We assume that actions that change an object property can be executed without being able to fully recognize the property itself. This is feasible in a simulated environment before deploying a robot in the real world. In a real world environment, where this assumption is more critical, we can use our method to improve agents' perception capabilities rather than learning them from scratch. Moreover, some actions can be executed without knowing the object properties, e.g., pushing a button to turn on a TV.

Acknowledgements

We thank the anonymous reviewers for their insightful comments. This work has been partially supported by AI-Plan4EU and TAILOR, two projects funded by the EU Horizon 2020 research and innovation program under GA n. 101016442 and n. 952215, respectively, and by MUR PRIN-2020 project RIPER (n. 20203FFYLK). We acknowledge the support of the PNRR project FAIR - Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU. This work has also been partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning STRIPS action models with classical planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, 399–407.

Aineto, D.; Jiménez, S.; and Onaindia, E. 2019. Learning action models with minimal observability. *Journal of Artificial Intelligence (AIJ)*, 275: 104 – 137.

Asai, M. 2019. Unsupervised grounding of plannable first-order logic representation from images. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 583–591.

Asai, M.; and Fukunaga, A. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Proceedings of the aaai conference on artificial intelligence*, volume 32.

Ashari, Z. E.; and Ghasemzadeh, H. 2019. Mindful active learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 2265–2271.

Bohg, J.; Hausman, K.; Sankaran, B.; Brock, O.; Kragic, D.; Schaal, S.; and Sukhatme, G. 2017. Interactive Perception: Leveraging Action in Perception and Perception in Action. *IEEE Transactions on Robotics*, 33: 1273–1291.

Bonet, B.; and Geffner, H. 2020. Learning First-Order Symbolic Representations for Planning from the Structure of the State Space. In *ECAI 2020*, 2322–2329. IOS Press.

Cakmak, M.; Chao, C.; and Thomaz, A. L. 2010. Designing interactions for robot active learners. *IEEE Transactions on Autonomous Mental Development*, 2(2): 108–118.

Cakmak, M.; and Thomaz, A. L. 2012. Designing robot learners that ask good questions. In *2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 17–24. IEEE.

Chao, C.; Cakmak, M.; and Thomaz, A. L. 2010. Transparent active learning for robots. In *2010 5th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 317–324. IEEE.

Coradeschi, S.; and Saffiotti, A. 2003. An introduction to the anchoring problem. *Robotics and autonomous systems*, 43(2-3): 85–96.

Cresswell, S.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using *LOCM*. *Knowledge Eng. Review*, 28(2): 195–213.

Dengler, N.; Zaenker, T.; Verdoja, F.; and Bennewitz, M. 2021. Online object-oriented semantic mapping and map updating. In *European Conference on Mobile Robots (ECMR)*, 1–7. IEEE.

Eppe, M.; Nguyen, P.; and Wermter, S. 2019. From semantics to execution: Integrating action planning with reinforcement learning for robotic tool use. *arXiv preprint arXiv:1905.09683*.

Gregory, P.; and Cresswell, S. 2015. Domain model acquisition in the presence of static relations in the LOP system. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 25, 97–105.

Hayes, B.; and Scassellati, B. 2014. Discovering task constraints through observation and active learning. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4442–4449. IEEE.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Hoffmann, J. 2001. FF: The fast-forward planning system. *AI magazine*, 22(3): 57–57.

Huang, S.-J.; Jin, R.; and Zhou, Z.-H. 2010. Active learning by querying informative and representative examples. *Advances in neural information processing systems*, 23.

Janner, M.; Levine, S.; Freeman, W. T.; Tenenbaum, J. B.; Finn, C.; and Wu, J. 2018. Reasoning about physical interactions with object-oriented prediction and planning. *arXiv preprint arXiv:1812.10972*.

Jocher, G.; Stoken, A.; Chaurasia, A.; and et al. 2021. ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support.

Juba, B.; Le, H. S.; and Stern, R. 2021. Safe Learning of Lifted Action Models. In Bienvenu, M.; Lakemeyer, G.; and Erdem, E., eds., *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021*, 379–389.

Kolve, E.; Mottaghi, R.; Han, W.; VanderBilt, E.; Weihs, L.; Herrasti, A.; Gordon, D.; Zhu, Y.; Gupta, A.; and Farhadi, A. 2017. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Pérez, T. 2018. From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *Journal of Artificial Intelligence Research (JAIR)*, 61: 215–289.

Kulick, J.; Toussaint, M.; Lang, T.; and Lopes, M. 2013. Active learning for teaching a robot grounded relational symbols. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, 1451–1457.

Lamanna, L.; Saetti, A.; Serafini, L.; Gerevini, A.; and Traverso, P. 2021. Online Learning of Action Models for PDDL Planning. In Zhou, Z., ed., *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, 4112–4118. ijcai.org.

Lamanna, L.; Serafini, L.; Saetti, A.; Gerevini, A.; and Traverso, P. 2022. Online grounding of symbolic planning domains in unknown environments. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, 511–521.

Liberman, A. O.; Bonet, B.; and Geffner, H. 2022. Learning First-Order Symbolic Planning Representations That Are Grounded. *CoRR*, abs/2204.11902.

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014. Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*, 740–755. Springer.

McCluskey, T. L.; Cresswell, S.; Richardson, N. E.; and West, M. M. 2009. Automated Acquisition of Action Knowledge. In *International Conference on Agents and Artificial Intelligence (ICAART)*.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C. A.; Ram, A.; Veloso, M.; Weld, D. S.; and Wilkins, D. E. 1998. PDDL—The Planning Domain Definition Language. Technical Report DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, Connecticut.

Migimatsu, T.; and Bohg, J. 2022. Grounding predicates through actions. In *2022 International Conference on Robotics and Automation (ICRA)*, 3498–3504. IEEE.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937. PMLR.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533.

Natale, L.; Metta, G.; and Sandini, G. 2004. Learning haptic representation of objects. In *International Conference on Intelligent Manipulation and Grasping*.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; and Antiga, L. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32: 8026–8037.

Petrick, R. P. A.; and Bacchus, F. 2002. A Knowledge-Based Approach to Planning with Incomplete Information and Sensing. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France*, 212–222. AAAI.

Pinto, L.; Gandhi, D.; Han, Y.; Park, Y.-L.; and Gupta, A. 2016. The curious robot: Learning visual representations via physical interactions. In *European Conference on Computer Vision (ECCV)*, 3–18. Springer.

Ribes, A.; Cerquides, J.; Demiris, Y.; and de Mántaras, R. L. 2015. Active learning of object and body models with time constraints on a humanoid robot. *IEEE Transactions on Cognitive and Developmental Systems*, 8(1): 26–41.

Saffiotti, A.; Broxvall, M.; Gritti, M.; LeBlanc, K.; Lundh, R.; Rashid, J.; Seo, B.-S.; and Cho, Y.-J. 2008. The PEIS-Ecology project: Vision and results. In *International Conference on Intelligent Robots and Systems (IROS)*, 2329–2335.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT Press.

Ugur, E.; and Piater, J. 2015. Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning. In *International Conference on Robotics and Automation (ICRA)*, 2627–2633. IEEE.