Refining discovered symbols with multi-step interaction experience

Emre Ugur and Justus Piater

***Abstract*— In our previous work, we showed how symbolic planning operators can be formed in the continuous perceptual space of a manipulator robot that explored the world with its single-step actions. In this paper, we extend our previous framework by enabling the robot to progressively update the previously learned concepts and rules in order to better deal with novel situations that appear during multi-step action executions. Our proposed system can infer categories of the novel objects based on previously learned rules, and form new object categories for these novel objects if their interaction characteristics and appearance do not match with the existing categories. Our system further learns probabilistic rules that predict the action effects and the next object states. There rules are automatically encoded in Planning Domain and Definition Language (PDDL), enabling use of powerful symbolic AI planners. Using this framework, our manipulator robot updated its reasoning skills from multi-step stack action executions. After learning, the robot was able to build stable towers in real world, exhibiting some interested reasoning capabilities such as stacking larger objects before smaller ones, and predicting that cups remain insertable even with other objects inside.**

I. Introduction

Autonomous robots require high-level grounded cognitive capabilities to achieve general-purpose complicated tasks. Hard-coding is not feasible, and conventional machine learning approaches will not work in high-dimensional, continuous perception-action spaces and with realistic amounts of training data. One way to get robots to learn higher-level concepts may be to focus on simple learning problems first, and then learn harder problems in ways that make use of simpler problems already-learned.

We followed such a path in our previous research [1], where, a manipulator robot built symbolic planning concepts and operators from it's own continuous interaction experience with the world. Starting from low-level object percepts, the robot organized its sensorimotor space forming object and effect categories; and learned logical rules that encode the relations between these categories in a form suitable for direct utilization of off-the-shelf AI planners. During progressive learning, the robot first explored the environment by executing actions on single objects, formed effect and object categories, gained the ability to detect object categories from its visual properties, and learned to predict the effects of actions. Next, with further interactions that involve pairs of objects, the robot learned more complex relations. These discovered symbols and rules could then be directly encoded in PDDL, enabling symbolic planning. With this, we argue that we closed the loop by going all the way from continuous sensorimotor experience to symbolic level, finally executing the plans in a real robot.

This research was supported by European Community's Seventh Framework Programme FP7/2007-2013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) under grant agreements no. 270273, Xperience, and no. 610532, Squirrel.

University of Innsbruck Institute of Computer Science, IIS Innsbruck, Austria firstname.lastname@uibk.ac.at

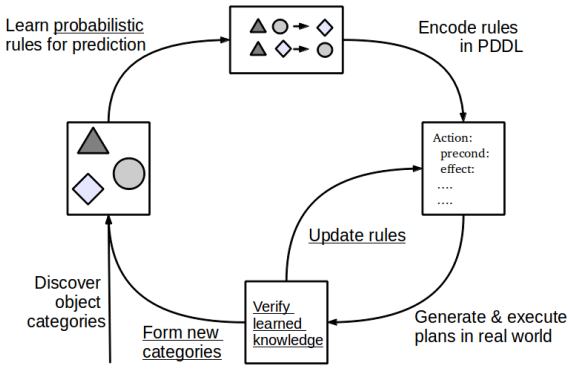


Fig. 1. The learning cycle. Adapted from the conceptual cycle defined in Xperience project: http://www.xperience.org/. The contributions in this paper are marked with underlined text.

In the system described above, the symbols and rules were learned from robot's isolated action executions. However, when the robot executes the sequence of actions planned towards a goal, it would encounter with completely novel situations that cannot be perceived or reasoned about with the knowledge that the robot acquired in previous learning. Therefore, the previously learned perceptual and prediction mechanisms should be updated based on robot's further experience obtained from different sequences of actions.

In our case, the robot previously learned operators from isolated stacking actions, where individual objects were stacked on top of other individual objects. After learning, we showed that the robot was able plan sequence of stack actions in order to build towers of arbitrary heights based on self-discovered object categories. However, the previously acquired knowledge would fail to capture the characteristics of a tower building task, which is quite different from stacking pairs of objects. Therefore, the robot is required to learn new concepts and rules related to the towers from its observation of sequential stack execution.

Fig. 1 provides an overview of our approach to progressive cognitive skill development. As we described above, our previous work went all the way from object category discovery to the execution of the symbolic plans, however it did not acquire further knowledge from the sequence of interactions. The contributions of this paper, which are marked in the figure, are as follows:

1) The categories of the novel complex objects, which are generated during interactions, are extracted from the learned rules, based on what kind of effects the objects generate in the following interactions. The previously learned rules are updated based on new information.
2) The novel objects are assimilated into the existing categories or new categories are accommodated for the novel objects depending on the visual properties of the novel objects. New object categories are formed if the visual appearance of components of this novel objects are not predicted to belong to the inferred category.
3) Probabilistic rules are learned from real-world interactions of objects, and symbolic planning is achieved using these learned operators.

Very few recent studies have addressed bottom-up construction of symbolic or sub-symbolic structures for planning in robotics. [2] and [3] clustered the continuous sensory space of the robot and generated multi-step plans in perceptual space with learned effect predictions. However, in these studies, the structures that were used for multi-step planning were still in continuous space, limiting the use of powerful AI planning techniques. Mugan and Kuipers [4] proposed a system that learns qualitative representations of states and predictive models in a bottom-up manner by discretizing the continuous variables of the environment. Konidaris et al. studied construction of symbols that are directly used as preconditions and effects of actions for generation of deterministic [5] and probabilistic [6] plans in interesting simulated environments. Our framework, on the other hand, learns non-linear relations between the discovered discrete symbols and the continuous percept of a manipulation robot in the real world.

II. Methods

In this section, we first give the manually designed perceptual and motor capabilities of the robot. In Section II-B, we summarize what kind of structures and rules were autonomously learned in and transferred from our previous study [1]. Finally, in Section II-C, we provide the main contribution of this paper, where the robot detects categories of novel objects, forms new categories, develops new prediction capabilities and probabilistic rules from observations of object interactions generated by sequence of actions.

A. Built-in knowledge

The robot is equipped with a number of manually-coded actions (a_j) that enable single- and multi-object manipulation. It can push a single object from different sides, pick-up, and release it; and also stack one object onto another. The single-object actions were used to find action-grounded object categories in previous stages, and not further explored in this paper.

The robot has the built-in functionality of detecting objects and extracting a number of visual features from them. The list of these features represent the objects in continuous sensory space and is denoted as $\boldsymbol{f}_o$ throughout the text. The continuous effect created in object features during action executions are also observed by the robot to find the discrete effect categories, as detailed in the next section.

B. Capabilities transferred from previous learning stages

In previous learning stages [1], the robot executed actions that involve single objects and pairs of objects, and progressively learned the following information. Note that throughout the text, the discovered categories are typed in uppercase.

- Effect space was discretized. For this, effect categories (ε_o^a) were formed by applying unsupervised clustering methods to the set of observed continuous effects for each action. The formed effect categories for different actions were as follows:

$$\begin{aligned} \varepsilon^{\text{pick-up}} &\in \{\text{GRASPED}\} \\ \varepsilon^{\text{release}} &\in \{\text{STABLE, TUMBLED}\} \\ \varepsilon^{\text{front-poke}} &\in \{\text{ROLLED, PUSHED}\} \\ \varepsilon^{\text{side-poke}} &\in \{\text{ROLLED, PUSHED}\} \\ \varepsilon^{\text{top-poke}} &\in \{\text{INSERTED, OBSTRUCTED}\} \\ \varepsilon^{\text{stack}} &\in \{\text{STACKED, INSERTED, TUMBLED}\}^1 \end{aligned} \tag{1}$$

- Action-grounded object categories ($\{c_o\}$) were formed. For this, objects that were affected similarly from the robot actions were grouped together. Therefore, object categories were encoded as the collection of effect categories generated by the five single-object actions ($c_o = (\varepsilon^{a_1}, \varepsilon^{a_2}, ..\varepsilon^{a_5})_o$). The formed object categories were as follows:

$$c_o \in \{\text{HOLLOW, SOLID, ROLLABLE, UNSTABLE}\} \tag{2}$$

  where

$$\begin{aligned} \text{HOLLOW} &= (\text{GRASPED, STABLE, PUSHED, PUSHED, INSERTED}) \\ \text{SOLID} &= (\text{GRASPED, STABLE, PUSHED, PUSHED, OBSTRUCTED}) \\ \text{ROLLABLE} &= (\text{GRASPED, STABLE, ROLLED, ROLLED, OBSTRUCTED}) \\ \text{UNSTABLE} &= (\text{-, TUMBLED, -, -, -}) \end{aligned}$$

- Prediction of object categories from their visual appearance was acquired. For this, the mapping from object features to the corresponding object categories was learned by training non-linear classifiers (SVMs):

$$c_o = \mathcal{C}(\boldsymbol{f}_o) \tag{3}$$

- Finally, predicting more complex action effects, i.e. effects of stack action, was learned. For this purpose, decision tree learners were trained to find logical rules that return the stack effect category given categories of the involved objects and their relations :

$$\varepsilon^{\text{stack}} = \mathcal{R}(c_b, c_r, rel(o_b, o_r)) \tag{4}$$

  Table I gives the results of decision tree learning along with the corresponding rules. We provide the set of previously learned rules in detail, as they will be used to infer the object categories in this paper.

[1]Note that originally more effect categories were formed for stack action, however further clustering was shown to generate the three final effect categories.

TABLE I

THE DETERMINISTIC RULES LEARNED FROM THE ROBOT SIMULATOR.

	Below = HOLLOW
	— Rel-Width = below-smaller
01	— — Above = HOLLOW: STACKED
02	— — Above = SOLID: STACKED
03	— — Above = ROLLABLE: INSERTED
04	— — Above = UNSTABLE: INSERTED
	— Rel-Width = same-width
05	— — Above = HOLLOW: STACKED
06	— — Above = SOLID: STACKED
07	— — Above = ROLLABLE: INSERTED
08	— — Above = UNSTABLE: INSERTED
09	— Rel-Width = below-bigger: INSERTED
	Below = SOLID
10	— Above = HOLLOW: STACKED
11	— Above = SOLID: STACKED
12	— Above = ROLLABLE: STACKED
13	— Above = UNSTABLE: TUMBLED
14	Below = ROLLABLE: TUMBLED

The knowledge summarized above is transferred to the system proposed in this paper for further development, as detailed in the next section.

C. Learning from sequence of actions

After learning logical rules on how to stack pairs of objects, the robot could build towers of arbitrary size using the available objects in the environment by planning and executing sequence of stacking actions. However, as the rules were learned from isolated single stacking interactions, the sequential effects of the successive action executions are not represented in these rules. In this section, we detail the methods that aim to enhance the learned rules and improve the planning capability by directly learning from the experience of sequence of stacking interactions, i.e. from building towers.

The learning is achieved in episodes that are composed of successive stacking interactions. Each episode starts with stacking two objects on top of each other, continues with putting new objects on top of the stack, and finishes when the object tower collapses. Fig. 2 shows one hypothetical episode, which lasts three interactions, where e_r and e_b correspond to the *released* and *base* entities, respectively. Note that the perception system cannot distinguish between touching objects, therefore the concept 'entity' will be used to refer to both individual and composite objects. If the entity e corresponds to an individual object o, than $e = o$, otherwise the entity is encoded as the list of the objects included: $e = \{o_0, o_1, ..\}$. While released entity (e_r) is always an individual object, the base entity (e_b) is an individual object only in the beginning of the episode. After the first interaction, new objects are added into e_b until the tower collapses.

The robot observes, computes, and stores the following information in each interaction i:

- Visual features of the detected entities before and after the interaction. The point clouds in beginning and at the end of the interaction are processed to compute visual features of the entities ($\boldsymbol{f}_{e_r^i}, \boldsymbol{f}_{e_b^i}$).

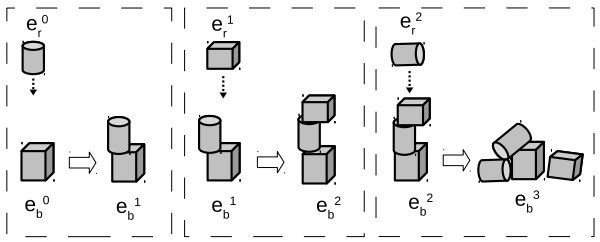


Fig. 2. One hypothetical episode used in learning from action execution sequence. This episode is composed of three interactions.

- Object category of the released entity, i.e. $c_{e_r^i}$. The classifier introduced in (3) is used to find the object category based on visual features of the object.
- Object category of the base entity, i.e. $c_{e_b^i}$. If this is the first interaction of the episode, $c_{e_b^i} = c_{o_b^i} = \mathcal{C}(\boldsymbol{f}_{o_b^i})$ (recall the classifier in (3)). If this is not the first interaction, i.e. the base entity is a composite object, the category of this entity is assigned to be unknown (-1), as the classifier was not trained with composite objects, therefore cannot make category predictions. One major aim of this paper is to reveal the categories of these composite objects by observing how they affect the subsequent stack interactions.
- The list of the objects that compose the base entity, formally:
$$e_b^i = \{o_r^{i-1}, \dots o_r^1, o_r^0, o_b^0\} \tag{5}$$
- The result of the stack interaction (ε^i). The result corresponds to the type of effect observed, and is assigned to one of the three effect categories ($\{$STACKED, INSERTED, or TUMBLED$\}$), which were provided in (1).

From above, one can notice that the interaction instances lack the information regarding the category of the entity on the ground (e_b), if this entity is a composite object. As shown in Fig. 2, entities on the ground are composed of several objects with different categories, and which category this composite object belongs to is ambiguous. As the objects might be inserted in others or remain on the top of the stack, the resulting category might be one of the categories that constitute the composite object. If the orientation of the released object change or the top surface is combination of several objects, the resulting category of the base object might be something completely different.

1) Inferring categories of complex entities: Categories of base entities are inferred using Algorithm 1. This algorithm finds the category of the base entity in each interaction (i), given the category of the released object (c_r^i), the effect observed in that interaction (ε^i), and the set of previously learned rules that predict the effect category given the categories of the objects and their relations (See (4) and Table I). The algorithm first checks if any possible interaction with one of the objects that is included in the base entity is represented in the set of rules (lines 3-7). These objects are checked starting from the last one as the latter added objects have higher probability to influence the category of the composite base entity. If no object satisfies the conditions

Algorithm 1 Inferring category of complex base entities

1: **for all** episodes **do**
2: **for all** interactions i **do**
3: **for all** objects $o' \in e_b^i$ **do**
4: **if** $\varepsilon^i = \mathcal{R}(c_{o'}, c_{e_r^i}, rel(o', e_r^i))$ **then**
5: $e_b^i \leftarrow o'$
6: assimilate $\boldsymbol{f}_{e_b^i}$ into c_r^i
7: **go to** 2
8: **for all** possible categories c_j **do**
9: **if** $\varepsilon^i = \mathcal{R}(c_j, c_{e_r^i}, rel(*, e_r^i))$ **then**
10: $c_{e_b^i} \leftarrow c_j$
11: **if** $\mathcal{C}(\boldsymbol{f}_{e_b^i}) = c_j$ **then**
12: assimilate $\boldsymbol{f}_{e_b^i}$ into c_j
13: **else**
14: accommodate $\boldsymbol{f}_{e_b^i}$ into c_j^{new}
15: **go to** 2

expressed in rules, then the system checks all possible object categories, even if they were not used previously in the current episode (lines 8-15). If the conditions of any rule is satisfied with this possible category, i.e. if the observed effect can be generated based on the rules with this category, then the base category is assumed to be this one. Please note that there is an important distinction between assigning a previous object to the base entity (line 5), and assigning a possible category (line 10). In the first case, the other features of the entity (such as width and height) are also known and stored, whereas in the latter one, the features of the base entity are left unknown.

2) Assimilation/accommodation mechanism: In Algorithm 1, we also described our assimilation and accommodation mechanism. The novel complex entity is assimilated into an existing category either if an object of the same category is a part of the complex entity (line 6) or the objects in the inferred category are visually similar to the complex entity (line 12). If these conditions are not satisfied, a new category is forked from the inferred category, and novel similar objects that are assigned to this new category.

3) Rule learning: After categories of base entities are inferred, the following dataset is formed:

$$\{c_{e_r^i}, c_{e_b^i}, rel(e_r^i, e_b^i), \varepsilon^i, c_{e_b^{i+1}}\}$$

Using this dataset, the system learns to predict the effect and the category of the formed entity given the categories of the released and base entities, and their relations:

$$(c_{e_r}, c_{e_b}, rel(e_r, e_b)) \rightarrow (\varepsilon, c_{e_b'}) \quad (6)$$

4) Symbolic planning: The robot builds symbolic domain and problem descriptions based on the object categories and the learned rules. This description, realized in the STRIPS notation, includes all the predicates and the actions. The predicates correspond to the automatically discovered object categories and relations. Actions correspond to the learned rules. The actions in PDDL contain the following three fields:

- Action name: We used stack action in this work.
- Preconditions: The list of the predicates that should be valid in order to apply the action. This corresponds to the object categories and their discrete relations (left part of (6)) for each learned rule.
- Effects: The list of the predicates that change if the preconditions are satisfied, and the action is executed. The predicted effect categories (right part of (6)) are provided in this field.

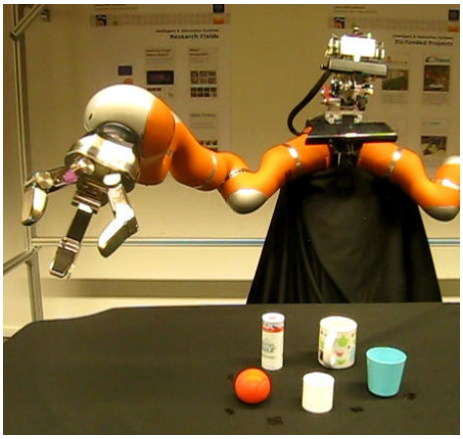


Fig. 3. The robot arm and gripper are used for manipulation, and Kinect is used to compute object features. Several objects are included while learning and verifying the planning capability in the real world.

Therefore, domain description includes a separate action for each learned rule along with the corresponding preconditions and effects. The initial state of the world, i.e. object categories and discrete relations between the objects, and the goal is defined in the problem description, in STRIPS notation as well. Given domain and problem descriptions, off-the-shelf symbolic planners are used to acquire the desired tasks.

The learned rules are represented probabilistically, however planning directly in probabilistic domain is inherently more complex compared to deterministic domain because of the high computation complexity [7]. In this paper, we take a path in between, compute deterministic plans using rules with highest probabilities, and re-plan with other rules if no plan is formed. Depending on the joint probability of the actions, the plan might or might not be executed - however we did not further analyze this verification step.

For plan generation, we use the Blackbox off-the-shelf planning software, which transforms the facts and operators defined in STRIPS notation into propositional satisfiability (SAT) problem and solves the problem with randomized systematic solvers.

III. Experiments

A. Robot Platform

Our experimental setup is composed of a KUKA Light Weight Robot arm and a Schunk gripper for manipulation, a Kinect sensor for environment perception, and a number of objects that are placed on the table for exploration (Fig. 3).

The workspace consists of several objects and a table, where the region of interest (ROF) is defined as the volume over the table. After extracting ROF from the point cloud of

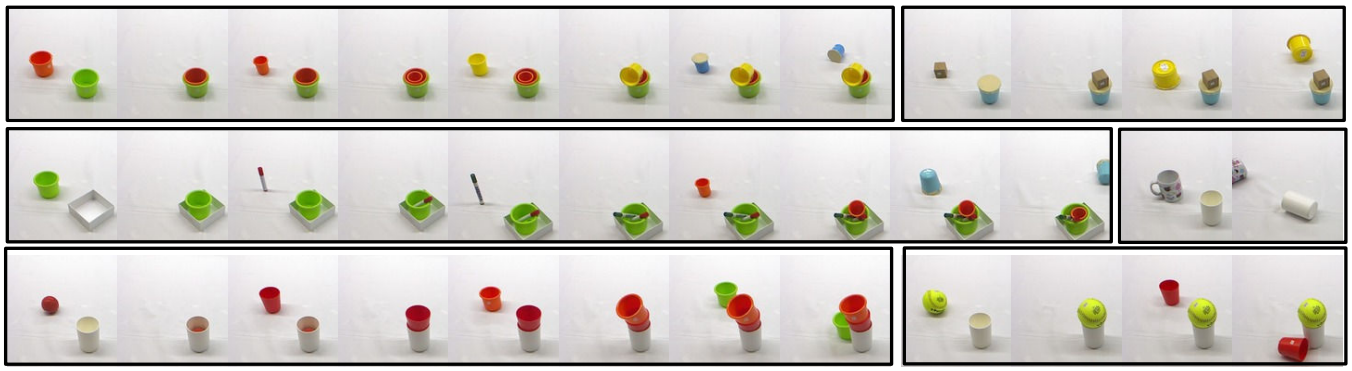


Fig. 4. Sample snapshots from stack interactions. Each image shows initial and final snapshot of of an interaction and each rectangle corresponds to one episode which ends when the tower collapses. The objects on the left are always stacked on top of the objects on the right.

Kinect, the objects are segmented by the Connected Component Labeling algorithm which differentiates object regions that are spatially separated. After segmentation, continuous object state is found by computing the following features:

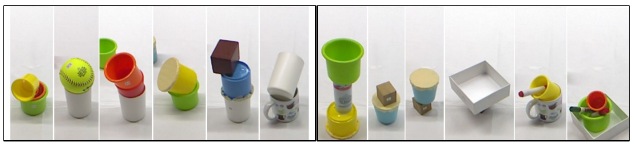
$$\boldsymbol{f}_o = (vis_o, pos_o, shape_o, dim_o, dist_o)$$

where vis feature encodes the knowledge regarding the existence of the object, $shape$ encodes the distribution of local surface normal vectors from the object surface. pos and dim correspond to the center and size of the object, respectively. Finally, $dist$ features encode the distribution of the local distance of all pixels to the neighboring pixels. See [1] for details.

The robot is equipped with a number of manually-coded actions that enable single- and multi-object manipulation. These actions are parameterized with target object position (pos_o), and the orientation of the gripper, depending on the action type. Once the target pose in task space is identified, the corresponding target joint angles are computed using inverse kinematics. Finally, given the current and target joint angles, a smooth trajectory is computed by the Reflexxes library, and this point-to-point movement is executed. In this paper, we use only the stack action, where the vertically-aligned gripper grasps an object using built-in spherical grip first, carries it on top of the another object, and releases it.

B. Interactions

In order to collect the interaction dataset, a human imitated stack action of the robot. The initial and final point clouds of each interaction are stored for later processing. The dataset contains 26 episodes and 66 interactions in total. A number of sample interactions are provided in Fig. 4. We did not use self-exploration of the robot in order to avoid damaging the plastic cover of the tactile sensors from large number of repeated object grasps. In order to imitate the noise in perception and the crude stacking skill of the robot, we introduced small offsets while dropping objects; and instead of carefully and gently placing the objects, we dropped the objects from the air similar to the release behavior of the robot. While ideally learning should be achieved through robot's own exploration, we discuss that it might be acceptable to make such simplifications in exploring object-object interactions as long as the learning results are verified with the real robot.



(a) Assimilate (b) Accommodate

Fig. 5. (a) Assimilate the entities into ROLLABLE category. (b) Accommodate a new category ROLLABLE-NEW for the entities.

C. Learned categories and category predictions

In this section, we analyze the learned rules that predict the next category of the base entity given the current categories of the base and released entities, and their relations.

First of all, the system formed a new object category, which has similar dynamics with ROLLABLE objects under stacking interactions, but different visual appearance. As shown in Fig. 5b, the entities represented by this new category generally correspond to towers of objects which are not stable or which do not allow further stacking. Not allowing further stacking is a similar characteristic with ROLLABLE objects, but as shown, the visual appearances of these entities are very different from ROLLABLE objects.

Table II provides the rules that are obtained from decision tree learning. In order to represent the collapsing towers, we manually add a new category, named UNDEFINED. The system revealed several different important underlying characteristic of the sequential stacking operations as follows:

- In a number of cases (e.g. rules 12-15), the next category of base entity is UNDEFINED, which corresponds to TUMBLED effect. UNDEFINED category is not further used as the episode ends with the TUMBLED effect.
- In some situations, the category of the released entity is transferred to the formed base entity. For example, if a SOLID entity is stacked on top of a bigger SOLID entity (rule 11) or on top of a HOLLOW entity (rule 04), a SOLID entity is formed with high probability. As another example, if a HOLLOW entity is stacked on top

TABLE II

THE PROBABILITIES OF GENERATING THE NEXT BASE ENTITY CATEGORY GIVEN INTERACTING ENTITIES. $(c_{e_r}, c_{e_b}, rel(e_r, e_b)) \rightarrow P(c_{e'_b})$

#	**Base**	**Released**	**Rel-Width**	HOLLOW	SOLID	ROLLABLE	UNSTABLE	ROLLABLE-NEW	UNDEFINED
01	HOLLOW	*	Base-smaller	0.33	0.00	0.33	0.00	0.00	0.33
02	HOLLOW	*	Same-width	0.00	0.50	0.50	0.00	0.00	0.50
03	HOLLOW	HOLLOW	Below-bigger	0.66	0.00	0.16	0.00	0.16	0.00
04	HOLLOW	SOLID	Below-bigger	0.00	1.00	0.00	0.00	0.00	0.00
05	HOLLOW	ROLLABLE	Below-bigger	0.60	0.00	0.40	0.00	0.00	0.00
06	HOLLOW	UNSTABLE	Below-bigger	0.50	0.00	0.25	0.00	0.25	0.00
07	SOLID	HOLLOW	Base-smaller	0.00	0.00	0.00	0.00	0.66	0.33
08	SOLID	HOLLOW	Below-bigger	0.66	0.00	0.33	0.00	0.00	0.00
09	SOLID	SOLID	Base-smaller	0.00	0.20	0.00	0.00	0.20	0.60
10	SOLID	SOLID	Same-width	0.00	0.00	1.00	0.00	0.00	0.60
11	SOLID	SOLID	Below-bigger	0.00	0.75	0.12	0.00	0.12	0.00
12	SOLID	ROLLABLE	*	0.00	0.00	0.00	0.00	0.00	1.00
13	SOLID	UNSTABLE	*	0.00	0.00	0.00	0.00	0.00	1.00
14	ROLLABLE	*	*	0.00	0.00	0.00	0.00	0.00	1.00
15	ROLLABLE-NEW	*	*	0.00	0.00	0.00	0.00	0.00	1.00

of a bigger SOLID entity (rule 08), the new formed base entity becomes HOLLOW with higher probability. These rules correspond to the situations where the released entity remains over the base entity after stacking.

- In some other situations, the category of the base entity is preserved. For example, when a small ROLLABLE object or an UNSTABLE object is stacked on top of a HOLLOW entity (rules 05 and 06), then base entity remains HOLLOW.
- As there is small number of interactions between objects with same-width, the corresponding learned rules are not very intuitive and generalizable (e.g. rule 02), therefore we will drop the same-width situations and assign strict size relations between objects in the rest of this paper.
- Finally, there are situations where the category of the generated base entity is not included in the set of categories of the objects that form this composite based entity. For example, when a HOLLOW object is stacked on top of a smaller SOLID object (rule 07), the category of the generated base entity becomes new ROLLABLE category. This result means that the generated entity behaves similar to ROLLABLE objects for stacking interactions, rather than they roll when poked from side. These situations are identified and learned in line 10 of Algorithm 1.

We argue that these rules reveal some interesting underlying characteristics of sequential stacking actions such as HOLLOW objects filled with small objects remain HOLLOW. However, these rules are neither inclusive nor all correct. For example, we can argue that rule 07 is not really intuitive: a HOLLOW object that is stacked on top of smaller SOLID object should generate a HOLLOW or UNDEFINED entity, but it generates ROLLABLE-NEW or UNDEFINED instead. When investigated in detail, this rule was extracted from instances, where these two objects are stacked on other HOLLOW objects, whose walls prevent the objects from falling, but also creating a structure which is not stackable anymore. We still believe that even the learned information is not perfect, it will enable construction of more safe and conservative plans. Rule 07, for example, will try to avoid stacking of large HOLLOW object on smaller SOLID objects, which would create unstable towers.

D. Learned effect prediction

The stack effects were previously predicted based on rules that were learned from set of isolated stack interactions performed in the robot simulation environment. However, the effects generated by sequential execution of stack actions in the real world is significantly different from the simulated ones, and this difference should be reflected in decision making and planning. For this purpose, we used C4.5 decision tree learning algorithm to find another compact set of rules with the corresponding probabilities from real interactions.

Table III provides the results of decision learning. Given given categories of released and base entities and their relative widths, each row gives the probability of generating an effect category by the stack interaction. The results can be interpreted as follows:

- (01-02): If the base entity is HOLLOW, than depending on the size of the released entity, different effects are generated. If the base entity has a larger width, then always an insertion occurs. However, if the base entity is smaller, then STACKED and TUMBLED also become possible. TUMBLED was not learned in the original rules

TABLE III

THE PROBABILITIES OF GENERATING AN EFFECT GIVEN OBJECT CATEGORIES AND THEIR RELATIONS. $(c_{e_r}, c_{e_b}, rel(e_r, e_b)) \rightarrow P(\varepsilon)$ STA., INS., TUM. REFER TO STACKED, INSERTED, TUMBLED EFFECTS.

#	**Base**	**Released**	**Rel-Width**	STA.	INS.	TUM
01	HOLLOW	*	Base-smaller	0.22	0.44	0.33
02	HOLLOW	*	Below-bigger	0.00	1.00	0.00
03	SOLID	HOLLOW	*	0.83	0.00	0.16
04	SOLID	SOLID	Base-smaller	0.40	0.00	0.60
05	SOLID	SOLID	Below-bigger	1.00	0.00	0.00
06	SOLID	ROLLABLE	*	0.00	0.00	1.00
07	SOLID	UNSTABLE	*	0.00	0.00	1.00
08	ROLLABLE	*	*	0.00	0.00	1.00
09	ROLL-NEW	*	*	0.00	0.00	1.00

TABLE IV

THE PREDICTED EFFECT AND ENTITY GIVEN GIVEN STACKED ENTITIES AND THEIR RELATIONS. $(c_{e_r}, c_{e_b}, rel(e_r, e_b)) \rightarrow (\varepsilon, c_{e'_b})$

	Below = HOLLOW
01	— Rel-Width = below-smaller: **TUMBLED (0.3), UNDEFINED (0.3)**
	— Rel-Width = below-bigger
02	— — Above = HOLLOW: **INSERTED (1.0), HOLLOW (0.6)**
03	— — Above = SOLID: **INSERTED (1.0), SOLID (1.0)**
04	— — Above = ROLLABLE: **INSERTED (1.0), HOLLOW (0.6)**
05	— — Above = UNSTABLE: **INSERTED (1.0), HOLLOW (0.5)**
	Below = SOLID
	— Above = HOLLOW:
06	— — Rel-Width = below-smaller: **STACKED (0.6), ROLLABLE-NEW (0.6)**
07	— — Rel-Width = below-bigger: **STACKED (1.0), HOLLOW (0.6)**
	— Above = SOLID:
08	— — Rel-Width = below-smaller: **TUMBLED (0.6), UNDEFINED (0.6)**
09	— — Rel-Width = below-bigger: **STACKED (1.0), SOLID (0.7)**
10	— Above = ROLLABLE: **TUMBLED (1.0), UNDEFINED (1.0)**
11	— Above = UNSTABLE: **TUMBLED (1.0), UNDEFINED (1.0)**
12	Below = ROLLABLE : **TUMBLED (1.0) , UNDEFINED (1.0)**
13	Below = ROLLABLE -new: **TUMBLED (1.0) , UNDEFINED (1.0)**

that had been trained in the robot simulator.

- (04-05): If both entities are SOLID, then they would be STACKED unless the released object is bigger. If the released object is bigger, the objects might be STACKED or TUMBLED with similar probabilities.
- (03): HOLLOW objects released over base objects would be probably STACKED, but there is small probably of observing TUMBLED effect. We expected a differentiation based on the relative sizes would appear, as in the previous bullet, but probably because of lack of representative interactions in training data, the system could not discover such rules.
- (06-07): The ROLLABLE or UNSTABLE entities that are released on SOLID objects generate TUMBLED effect. In the rules learned from the robot simulator, the ROLLABLE objects was creating STACKED effect when released on the SOLID objects, however in reality we observed that they roll off the objects after released.
- (08-09): If the entity on the base is a ROLLABLE one or belongs to the new ROLLABLE-NEW category, which corresponds to unstable towers, then the effect is always TUMBLED.

The set of rules above (Table III) make more realistic predictions in the real world compared to the set of rules learned from the robot simulator (Table I). Yet, in simple cases, such as releasing small objects on big HOLLOW objects or releasing any object on ROLLABLE objects, the predicted effects are consistent among these tables.

E. Generated STRIPS rules

In STRIPS notation, the postcondition/effect field should include both the effect of the action and the category of the formed base object. Therefore, the system should learn compact rules that encode both predictions simultaneously. We again used C4.5 decision tree learners and obtained the rules provided in Table IV. The PDDL domain description is automatically generated from this set of rules.

An explanatory sample action that corresponds to rule 04 is provided in Fig. 6. Lines 03-05 state preconditions and 07-12 state effects of the action. Predicate 'U', which is

```
01 (:action stack ;;; rule no: 04
02 :parameters (?Below ?Above)
03 :precondition (and (Hollow ?Below)
04 (Rollable ?Above) (Below-bigger ?Below ?Above)
05 (U0 ?Above) (U2 ?Below) (N2) (H0) )
06 :effect
07 (not (U0 ?Above))
08 (U3 ?Above) (U2 ?Below)
09 (N3) (not (N2))
10 (not (Hollow ?Below))
11 (not (Rollable ?Above))
12 (Hollow ?Above))
```

Fig. 6. Automatically generated sample action in PDDL that corresponds to rule 04 in Table IV. (?Above, ?Below) pair refer to the released and base entities in precondition fields. ?Above variable starts representing the formed base entity in the effect field.

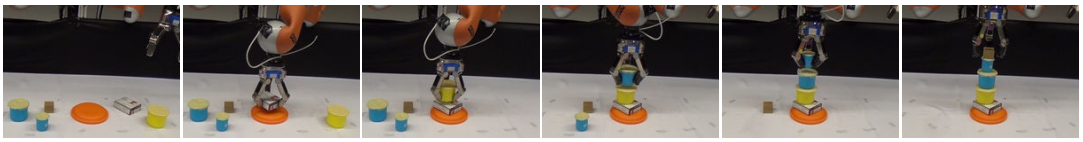
'U0' for all objects in the beginning, encodes the order of the object in stacking. 'N' and 'H' predicates correspond to the number of objects in the stack and height of the stack, and they change based on the predicted effect. For example, while INSERTED effect only increments 'N' predicate, STACKED effect increments both 'N' and 'H' predicates. As the increment operator is not supported in STRIPS notation, a separate action is automatically generated for each level of H and S's. The category of Above object (which takes the role of base entity) is assigned to 'Hollow' (line 12). In order to remove the Below object from the planning world (as Below and Above are combined now), the category of Below object is assigned to '(not Hollow)'.

F. Real World Experiments

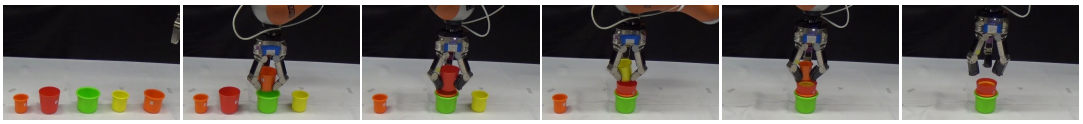
In this section, we provide the results obtained from real robot experiments, and discuss a number of points observed during perception, planning and execution of the robot. The learned representations and rules were already provided in the previous sections, therefore this section is more about verifying the learned knowledge in representative real-world scenarios, and discussing the capabilities and limitations of our proposed method within these scenarios.

Fig. 7 provides snapshots from a number of plans generated and (blindly) executed for the goal of building towers that include all the objects independent of any constraint on compactness or height. As seen, given various objects with different affordances, the robot planned the sequence of actions that generate stable towers. In these plans, always smaller objects were inserted in or stacked on larger objects. In other words, due to the the learned probabilistic rules that favor stacking small objects on larger objects (e.g. rules 04 and 05 in Table III), the robot implicitly learned building stable towers from objects of different sizes. Note that this knowledge was not encoded in the rules that were transferred from previous stages (Table I), which shows the advantages of further learning from real interactions. From Fig. 7c, one can also observe that the robot correctly reasoned about the properties of the containers by planning and attempting to insert several board-markers (detected as UNSTABLE objects) into the container (detected as HOLLOW object).

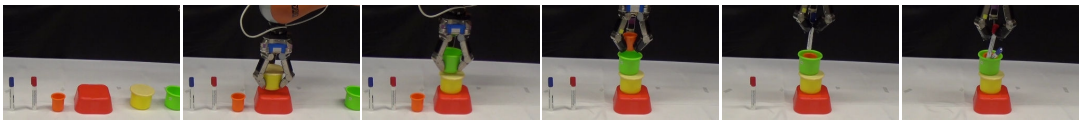
During the experiments, we also observed sources for failures in building towers. First of all, the robot sometimes



(a) Tower of blocks



(b) Stack of cups



(c) Stack of objects with mixed categories

Fig. 7. Snapshots from a number of plan generation and executions. In all cases, the robot plans compact and stable towers with the available objects. (a) and (b) shows the robot builds the stacks starting from larger objects, and (c) shows that the robot can reason about inserting several small objects into the containers. In PDDL, goals are set as 'N6', 'N5', and 'N6', respectively. All the data used in learning and predictions, along with the robot execution videos are available at http://emreugur.net/humanoids2015/.

suffered from incorrect categorization of objects based on visual features obtained from Kinect, therefore could not make correct plans. The small cube in Fig. 7a was one of such objects, which was sometimes categorized as a SOLID object and sometimes ROLLABLE one. The second reason for the failures was the inaccuracies in perception of the release location and execution of the stack action. Finally, there were some new categories that do not conform to any extracted rules, therefore not learned by the system. For example, if a long UNSTABLE object is inserted into a HOLLOW object (as in last snapshot of Fig. 7c), the generated new object behaves like a HOLLOW object for small objects, like a SOLID object for some other objects, and like a ROLLABLE object, causing TUMBLED effect, for larger objects. The system is required to learn new categories that are combination of the existing ones.

We argue that the problems addressed above can be avoided by better perception, more training, and more advanced action representations. However, there are other more major limitations which are not straightforward to address with the our current approach. Let us focus the last snapshot of Fig. 7c. Here the system reasons that HOLLOW objects remain HOLLOW if smaller or UNSTABLE objects are inserted in. However, after successive insertions, depending on the relative sizes of the objects, the HOLLOW object will be completely filled with the small objects, and cannot not be categorized as HOLLOW anymore. While our system provides the capability to detect whether the objects remain HOLLOW or not from their visual perception during action executions; it does not have the capability to reason about how many objects can fit into a HOLLOW objects. This kind of reasoning probably requires other concepts (such as volume of the hole instead of width and height of the object) and learning of more complex capabilities such as arithmetic processing over learned concepts.

IV. Conclusion

In this paper, we showed how the robot can further develop previously learned concepts and reasoning skills in order to better represent multi-step interaction characteristics and to make better plans. The updated skills were verified in real-world settings in a tower building scenario where the robot was shown to successfully generate and execute plans that form stable towers. However, as we partially discussed at the end of the previous section, our system is currently limited to predicting the next state based on small number of object categories of only the current state and very simple object relation information. In future, we plan to investigate how the robot can discover other task dependent and potentially hidden concepts and variables such as stability and height of the towers, volume of the containers, and how it can learn more challenging rules that can make reasoning with higher-level knowledge.

References

[1] E. Ugur and J. Piater, "Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning," in *ICRA*, 2015, pp. 2627–2633.

[2] E. Ugur, E. Oztop, and E. Sahin, "Goal emulation and planning in perceptual space using learned affordances," *Robotics and Autonomous Systems*, vol. 59, no. 7–8, pp. 580–595, 2011.

[3] J. Pisokas and U. Nehmzow, "Experiments in subsymbolic action planning with mobile robots," in *Adaptive Agents and Multi-Agent Systems II, Lecture Notes in AI*. Springer, 2005, pp. 80–87.

[4] J. Mugan and B. Kuipers, "Autonomous learning of high-level states and actions in continuous environments," *Autonomous Mental Development, IEEE Transactions on*, vol. 4, no. 1, pp. 70–86, 2012.

[5] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "Constructing symbolic representations for high-level planning," in *28th AAAI Conf. on AI*, 2014.

[6] G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "Symbol acquisition for probabilistic high-level planning," in *International Joint Conference on Artificial Intelligence*, 2015.

[7] A. L. Blum and J. C. Langford, "Probabilistic planning in the graphplan framework," in *Recent Advances in AI Planning*. Springer, 2000, pp. 319–332.