Bottom-Up Learning of Object Categories, Action Effects and Logical Rules: From Continuous Manipulative Exploration to Symbolic Planning

Emre Ugur and Justus Piater

Abstract— **This work aims for bottom-up and autonomous development of symbolic planning operators from continuous interaction experience of a manipulator robot that explores the environment using its action repertoire. Development of the symbolic knowledge is achieved in two stages. In the first stage, the robot explores the environment by executing actions on single objects, forms effect and object categories, and gains the ability to predict the object/effect categories from the visual properties of the objects by learning the nonlinear and complex relations among them. In the next stage, with further interactions that involve stacking actions on pairs of objects, the system learns logical high-level rules that return a stacking-effect category given the categories of the involved objects and the discrete relations between them. Finally, these categories and rules are encoded in Planning Domain Definition Language (PDDL), enabling symbolic planning. We realized our method by learning the categories and rules in a physics-based simulator. The learned symbols and operators are verified by generating and executing non-trivial symbolic plans on the real robot in a tower building task.**

I. Introduction

There exists a representational gap between the continuous sensorimotor world of a robot and the discrete symbols used by advanced AI planning methods. One approach for bridging this gap is to learn the mapping between the symbols and the sensorimotor readings of the robot. The studies that follow this approach typically assume the existence of pre-coded planning symbols, and investigate how to learn the relations between these pre-coded symbols and continuous world of the robot [1]. They generally define transition rules that are linked by logical preconditions and postconditions, and use the sensorimotor experience of the robot to associate the predicates of the preconditions and postconditions to the robot percepts [2], [3].

On the other hand, Sun argued that symbols "are not formed in isolation", and "they are formed in relation to the experience of agents, through their perceptual/motor apparatuses, in their world and linked to their goals and actions" [4]. In this vein, symbol formation in a robot interacting with its world was investigated in a number of studies. In [5], self-organizing maps were used to cluster the low-level sensory data of a mobile robot. Each cluster was assigned to a new perceptual state, and planning was achieved by successively predicting the next perceptual states. Our previous work [6] also addresses planning in perceptual space, where the affordances that the robot learned during its interactions with the environment were used to develop multi-step plans in perceptual space with effect predictions and forward chaining in continuous domain. 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. [7], on the other hand, took a path in between, and used a teacher to learn grounded relational non-predefined symbols.

Very few recent studies have addressed bottom-up construction of symbolic structures for planning in robotics. Mugan and Kuipers [8] 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. Based on the predictive models that are learned using Dynamic Bayesian Networks (DBNs), the Markov decision process (MDP) framework is used to plan goal-oriented hand/arm control in the simulator. Konidaris et al. [9] studied construction of symbols that are directly used as preconditions and effects of actions. PDDL is automatically generated from these symbols, enabling high-level planning in a simulated 2d environment with a rich set of actions and effects. Our system, different from the ones above, can learn non-linear and highly-complex relations between the discovered discrete symbols and the continuous world of the robot in real world settings.

The current study is part of a research effort where a robot system gradually develops skills and competencies in subsequent stages of development, similar to human infants. Previously we showed that a robot that is initialized with a basic reach-and-grasp movement capability can discover a set of action primitives, learn a library of affordances and associated predictors, and finally use these structures to bootstrap complex imitation with the help of a cooperative tutor [10]. The methods provided in the current study enable the robot to reach a higher level of cognitive competence, where it forms structures for planning at a symbolic level.

II. Methods

A. Representation

The robot is equipped with a number of manually-coded **actions** that enable single- and multi-object manipulation. The robot can poke a single object from different sides using front-poke (*f-poke*), side-poke (*s-poke*), and top-poke

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 agreement no. 270273, Xperience.

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

(*t-poke*) actions; grasp it using the *pick* action, and drop it at a given position using the *release* action. It can also stack one object onto another using the *stack* action, which is a combination of *pick* and *release*, where the vertically-aligned gripper grasps the object first, carries it on top of the other, and releases it.

Continuous object state is represented by $\boldsymbol{f}_o$, and includes the list of object features obtained from the visual perception, including features related to the existence, shape and size of the object. The continuous proprioceptive state of the robot, which includes the gripper position, is represented by $\boldsymbol{f}_r$. Thus, the **continuous world state** with n objects is represented by

$$(\boldsymbol{f}_{o_1}, \boldsymbol{f}_{o_1}, \dots, \boldsymbol{f}_{o_n}, \boldsymbol{f}_r)$$

Continuous effect of an action on an object is represented by $\Delta(\boldsymbol{f}_o, \boldsymbol{f}_r)^a$, and corresponds to the changes in object features and proprioceptive readings. **Effect category** (ε_o^a), on the other hand, is discovered by the robot during its development.

An object can categorically be represented as the list of action possibilities it offers to the robot. For example, a mug can be represented as a list of action effects including 'up' effect when lifted, 'reduce-weight' effect when poured, etc. Therefore, **object category** corresponds the collection of effect categories that are expected to be obtained by the m available actions:

$$S_o = (\varepsilon^{a_1}, \varepsilon^{a_2}, ..\varepsilon^{a_m})_o \tag{1}$$

where each unique combination of the effects is assigned to a different index.

Finally, **discrete world state** is represented by the collection of object categories and discrete relations (R) between the objects:

$$S = (S_{o_1}, S_{o_2}, \dots, S_{o_n}, R1_{o_1,o_2}, R2_{o_1,o_2}, \dots)$$

Note that the discrete relations are pre-defined in the current study.

B. Development of symbols and prediction capability

The robot explores the environment using its action repertoire, and encodes its observations in the form of continuous world states and effects. While exploring the environment, the robot progressively acquires the ability to represent the world with learned symbolic predicates, and to predict effects of its own actions based on learned logical rules. These predicates and rules are later used for symbolic planning. The stages of development are as follows:

1) Stage I: Learning object categories: In the first stage, the robot explores the environment with its actions that are executed on single objects such as poke, pick and release. It observes and stores the continuous effects generated during these interactions. After experiencing N^a number of interactions for any action a, unsupervised clustering methods are applied in continuous effect space

$$\{\varepsilon_i^a\}_{i=1}^I = \text{Cluster}(\{\Delta(\boldsymbol{f}_o, \boldsymbol{f}_r)_j^a\}_{j=1}^{N^a}), \tag{2}$$

in order to find I effect categories for each action. Recall that in (1) we defined an object category as a collection of the effect categories. Thus, through this clustering process, the robot learns the object categories it encountered in terms of the action effects.

Next, the robot learns to relate these categories to the features of the objects in order to detect object categories without any action execution. For this purpose, it learns the mapping from continuous object features to the effect categories by training multi-category classifiers (Predictor$^{a_i}(\boldsymbol{f}_o)$) for each action a_i, with the following training data:

$$\{(\boldsymbol{f}_o, \boldsymbol{f}_r)_j, \varepsilon_j^a\}_{j=1}^{N^a} \tag{3}$$

After learning, given object features, the robot can predict the list of offered effect categories, i.e. the object category:

$$S_o = (\text{Predictor}^{a_1}(\boldsymbol{f}_o), ...\text{Predictor}^{a_m}(\boldsymbol{f}_o)) \tag{4}$$

For instance, if one object is predicted to be rolled with a poke action, lifted with pick action, and tumbled-off when it is released, it is categorized as (rolled, lifted, and unstable) tuple. At the end of this phase, the robot acquires the ability to categorize the observed objects with the type of effects its action repertoire can generate.

2) Stage II.a: Discovering the effect categories of stack action: In this stage, the robot explores the environment with its *stack* action that involves pairs of objects. Similar to the previous stage, the robot executes this action on different pairs of objects, observes the continuous effects generated, and finds representative and meaningful effect categories created on the pairs.

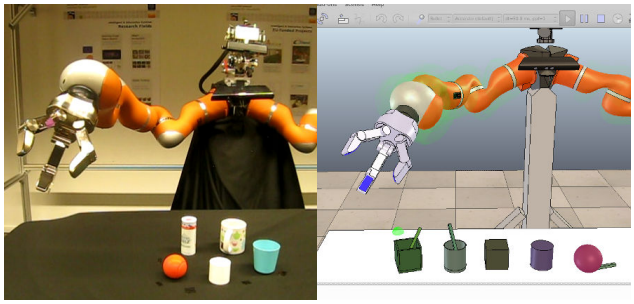
For the effect category generation, we observed that a naive clustering in continuous effect space was not effective as the interacting objects can generate various effects due to the complex interactions between them. Applying further exploratory actions on the objects after stacking might help the system to handle this complexity. For example, after a stack action, if the robot grasps the object below, lifts and rotates it, the stack relation between those objects will be more clearly seperable in the effect space (e.g. if they move together then they are properly-inserted). In the current work, the poke action is used as the exploratory action. After stacking, the robot pokes both of the objects one by one, observes the additional generated effects, and includes all these data to find a grouping in interactions:

$$\begin{aligned}\{\varepsilon_i^{\text{stack}}\}_{i=1}^I = \text{Cluster}(\{&\Delta(\boldsymbol{f}_{o_1}, \boldsymbol{f}_{o_2}, \boldsymbol{f}_r)^{a_{\text{stack}}}, \\ &\Delta(\boldsymbol{f}_{o_1}, \boldsymbol{f}_{o_2}, \boldsymbol{f}_r)^{a_{\text{f-poke},o_1}}, \Delta(\boldsymbol{f}_{o_1}, \boldsymbol{f}_{o_2}, \boldsymbol{f}_r)^{a_{\text{f-poke},o_2}}\})\end{aligned} \tag{5}$$

where $a_{\text{f-poke},o_i}$ corresponds to the front-poke action that is applied to the object pair (o_1, o_2). The order of pairing is important, and the indexes will be replaced with the 'below' and 'above' keywords, in the rest of the text.

3) Stage II.b: Learning logical rules for stack: In this stage, the robot builds rules that encode the effects of the stack action. It uses decision tree rule learning methods to build a decision tree for a compact set of rules using the object categories and the relations between the objects:

$$\{(S_{o_1}, S_{o_2}, R1_{o_1,o_2}, R2_{o_1,o_2}) \rightarrow \varepsilon^{a_{\text{stack}}}\} \tag{6}$$



(a) Real robot (b) Simulation

Fig. 1. The robot arm and gripper are used for manipulation, and Kinect is used to compute object features. The simulation environment includes one and two objects during single-object and paired-object affordance learning experiments, respectively. Several objects are included while testing the planning capability for tower-building task in the real world.

where $R1$ and $R2$ represent discrete relations between objects, and ε^{a_s} is the effect category obtained by the stack action, that was found in (5).

4) Stage III: Symbolic planning: In this stage, the robot builds symbolic domain and problem descriptions based on the object states and the rules learned in the previous stages. 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.

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. Please see Section III-E for the implementation details.

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. 1a).

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 Kinect, the objects are segmented by the Connected Component Labeling algorithm which differentiates object regions that are spatially separated by a preset threshold value (3 cm in the current implementation). 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. $shape$ features are encoded as the distribution of local surface normal vectors from object surface. Specifically histograms of normal vectors along each axis, 8 bins each, are computed to form $3\times8 = 24$ sized feature vector (see [6] for details). 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. For this purpose, for each pixel we computed distances to the neighboring pixels along each 4 direction on Kinect's 2D depth image. For each direction, we created a histogram of 20 bins with bin size of $0.5cm$, obtaining a $4 \times 20 = 100$ sized vector for the $dist$.

The robot can manipulate objects using its action repertoire that includes pick, release, f-poke, t-poke, s-poke, and stack actions. 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. functions provided by the and Dynamics Library. 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. The object is gripped from the top using the built-in spherical grip of the Schunk hand.

For learning, the robot is required to make large number of interactions with the objects, which is time-consuming and risky in the real world. For this purpose, we used the V-Rep robot simulation platform (Fig. 1b) with the Bullet physics engine. The objects used in training include boxes, cylinders, spheres, box walls, cylinder walls as shown in Fig. 1b.

B. Learned object categories

In the first stage of learning, the simulated robot executes *poke*, *pick*, and *release* actions on single objects, monitors the environment, and stores the continuous object states along with the continuous effects generated by different actions. After collecting the interaction instances

TABLE I

Discovered effect categories for single-object actions.

Action	Effect prototype	ε^{a_i}	Interpretation
Pick	-	0	Grasped
Release	change in object position no change	0 1	Tumbled Stable
Front-poke	change in object visibility no change	0 1	Rolled off the table Pushed
Side-poke	change in object visibility no change	0 1	Rolled off the table Pushed
Top-poke	large change in gripper pos small change in gripper pos	0 1	Finger goes through Finger obstructed

TABLE II

DISCOVERED EFFECT CATEGORIES OF STACK ACTION. NUMBER OF OCCURRENCES OF THE CLUSTERS, THE CORRESPONDING CHANGES IN OBJECT FEATURES, AND THEIR INTERPRETATION ARE GIVEN.

	After $stack_{o_1,o_2}$		After *f-poke*$_{o_1}$				After *f-poke*$_{o_2}$				Interpretation
#	Δvis_{o_1}	Δvis_{o_2}	Δpos_{o_1}	Δpos_{o_2}	Δvis_{o_1}	Δvis_{o_2}	Δpos_{o_1}	Δpos_{o_2}	Δvis_{o_1}	Δvis_{o_2}	
49	1	1	1	1	1	1	1	1	1	1	INSERTED
127	1	1	1	1	1	0	1	1	1	1	STACKED1
28	1	1	1	1	1	0	0	1	1	1	STACKED2
67	1	1	0	0	0	1	0	1	0	1	TUMBLED1
13	1	1	0	1	0	1	0	1	0	1	TUMBLED2
6	1	1	1	1	1	1	0	0	1	1	-
4	1	1	0	0	0	0	0	0	0	0	-
..	..	..	..	..	..	..	..	..	..	..	..

$\{\boldsymbol{f}_o, \boldsymbol{f}_r, \Delta(\boldsymbol{f}_o, \boldsymbol{f}_r)^{a_j}, a_j\}$, it first applies clustering in the effect space, $\{\Delta(\boldsymbol{f}_o, \boldsymbol{f}_r)^{a_j}\}$, and finds a number of effect categories for each different action. We applied clustering in object position, object visibility and gripper position spaces, and obtained the effect categories presented in Table I. The effect categories ε^{a_i} are indexed seperately for each different action. Note that a stable object that is released on the table does not tumble, so no change in its position occurs.

As all the objects used in the experiments were graspable, the effect of applying *pick* action were always same. For other actions, different distinguishable effects were created in different spaces. When these effects are enumerated, the following compact object category representation is obtained:

$$S_o \in \{\text{SOLID}, \text{ROLLABLE}, \text{HOLLOW}, \text{UNSTABLE}\}$$

where

SOLID = (0, 1, 1, 1, 1) ROLLABLE = (0, 1, 0, 0, 1)
HOLLOW = (0, 1, 1, 1, 0) UNSTABLE = (-, 0, -, -, -)

The effect categories for UNSTABLE objects are unknown, as they are generally thin objects that lie on the ground, risky to interact with.

Finally, the mapping from object features to object categories, i.e. $\boldsymbol{f}_o \rightarrow S_o$, is learned using a Support Vector Machine (SVM) with the Radial Basis Function (RBF) kernel and optimized parameters [11]. Note that SVMs mainly used $shape$ and $dist$ features to predict object categories, but further analysis is not within the scope of this paper.

C. Discovered stack effect categories

In this stage, the simulated robot executes *stack* action on pairs of objects, monitors the environment, and stores the continuous object states along with the continuous effects generated by different actions. As we described before, the effects do not only correspond to the immediate effect of the *stack* action, but is a collection of effects generated by the following exploratory *front-poke* actions applied to both of the objects. Furthermore, as the effect space was very complex, only the changes in the visibility and position of the object are used as the effect features. The grouped instances of these features are provided in Table II. We interpreted these effect categories based on position and visibility changes, and provided labels for each effect category in the table. STACKED1 and STACKED2 effect categories are generated by SOLID-on-SOLID and ROLLABLE-on-SOLID stacking in general. TUMBLED1 and TUMBLED2 effect categories on the other hand are created by SOLID-on-ROLLABLE stacking interactions, where in the first case poking one object does not affect the other, and in the second case, the poked ROLLABLE object also pushes the SOLID object. The labels of the effect categories will be used in the rest of the text (similar to use of object category labels) solely to ease the understandability of the text.

TABLE III

K-MEANS CLUSTERING IN EFFECT CATEGORY SPACE.

ε^{stack}	μ_1	σ_1	μ_2	σ_2	Cat.
INSERTED	[0,0,0]	[0,0,-9]	[0, 2, 0]	[2, 4, 4]	0
STACKED1	[0,0,0]	[1,0,-2]	[1, 1, 0]	[4, 3, 5]	1
STACKED2	[0,0,0]	[1,-1,-1]	[0, 1, 0]	[4, 4, 2]	1
TUMBLED1	[2,0,0]	[0,-9,-12]	[3, 3, 0]	[7, 11, 4]	2
TUMBLED2	[-1,0,0]	[1,-5,-16]	[3, 2, 0]	[9, 13, 3]	2

If we assume that the number of meaningful effect categories is 3, and apply further clustering (K-Means) using mean and variance of the object position changes for each effect category, a more 'intuitive' categorization can be obtained at the end as shown in Table III. As seen in the table, STACKED1 and STACKED2 are assigned to a single category; and TUMBLED1 and TUMBLED2 are assigned to another category. While these results with a hierarchical clustering give some insights about the complexity of the problem, and the necessity for human intervention to obtain the most meaningful clusters, we are not going to use these clusters. Instead, the most occuring five effect categories (Table II) will be transferred to the next stage.

D. Rule Learning

Based on the discovered object categories and the discovered effect categories, now the robot can represent the interactions with discrete variables. Here, to represent discrete world state, we use discovered object categories, and size and height relations. The set of interactions the robot observed is encoded as

$$\{(S_{o_1}, S_{o_2}, \text{Rel-Width}_{o_1,o_2}, \text{Rel-Height}_{o_1,o_2}), (\varepsilon^{stack})\}$$

where

$$S_o = \{\text{SOLID, ROLLABLE, HOLLOW, TUMBLED}\}$$
$$\text{Rel-Width} = \{\text{below-bigger, is-same, below-smaller}\}$$
$$\text{Rel-Height} = \{\text{below-higher, is-same, below-shorter}\}$$
$$\varepsilon^{stack} = \{\text{INSERTED, STACKED1, STACKED2, TUMBLED1, TUMBLED2}\}$$

A number of sample interaction instances obtained during experiments and encoded in the discovered discrete structures are shown below, where the first and second objects corresponds to the object below and above during stacking.

'HOLLOW','HOLLOW','below-smaller','below-shorter','STACKED1'
'HOLLOW','HOLLOW','below-bigger','below-shorter','INSERTED'
'HOLLOW','SOLID','same-width','below-higher','STACKED1'
'SOLID','HOLLOW','below-smaller','below-shorter','STACKED1'
'SOLID','ROLLABLE','below-smaller','same-height','STACKED2'
'ROLLABLE','SOLID','below-bigger','same-height','TUMBLED1'

C4.5 decision tree learning with pruning is used to find a compact representation of rules with the Weka software package. The obtained decision tree is presented in Fig. 2. As shown, if the object below is HOLLOW, then depending on the size of the object above, the effects are different. For example, if the object below is bigger, then the resulting effect is always INSERTED, independently of any other factor. Otherwise, depending on the category of the above object and its relative width, it can get INSERTED into or stacked onto the HOLLOW object. If the object below is SOLID, then, again depending on the object above, different types of stacked effects are expected to be generated. Note that in both STACKED1 and STACKED2, the above object stacks on the below object, but the STACKED2 case is more UNSTABLE (as detected and learned by the successive exploratory poke actions). Finally according to the decision tree, if the below object is ROLLABLE or UNSTABLE, independent of the object above, the effect would be tumbled or stacked, respectively. The tumbled effect in response to a stack action on ROLLABLE objects was expected. However, the stacked effect in response to stack action on UNSTABLE objects is not intuitive, and needs more elaborate analysis of the decision tree learner. Note that the rule learner was able to discover that the relative height of the objects do not have any significant influence on the generated effects in the simulated interactions; thus height relation is not included in any learned rule.

	Below = HOLLOW
	— Rel-Width = below-smaller
01	— — Above = HOLLOW: STACKED1
02	— — Above = SOLID: STACKED1
03	— — Above = ROLLABLE: INSERTED
04	— — Above = UNSTABLE: INSERTED
	— Rel-Width = same-width
05	— — Above = HOLLOW: STACKED1
06	— — Above = SOLID: STACKED1
07	— — Above = ROLLABLE: INSERTED
08	— — Above = UNSTABLE: INSERTED
09	— Rel-Width = below-bigger: INSERTED
	Below = SOLID
10	— Above = HOLLOW: STACKED1
11	— Above = SOLID: STACKED1
12	— Above = ROLLABLE: STACKED2
13	— Above = UNSTABLE: STACKED1
14	Below = ROLLABLE: TUMBLED1
15	Below = UNSTABLE: STACKED1

Fig. 2. Results of decision tree learning. For example the rule 01 states that if the object below is HOLLOW, and it is smaller than a HOLLOW object that is dropped, then STACKED1effect is predicted to occur.

```
(define (domain stack)
(:predicates                (hollow ?x)
(solid ?x)                  (below-smaller ?x ?y)
(rollable ?x)               (below-bigger ?x ?y)
(unstable ?x)               (same-size ?x ?y)
(pickloc ?x)                (below-shorter ?x ?y)
(stackloc ?x)               (below-higher ?x ?y)
(instack ?x))               (same-height ?x ?y)
```

Fig. 3. Predicates that are used in domain definition.

E. Planning

Based on the rules learned in the previous section, the domain definition can be automatically constructed in STRIPS notation using the learned predicates (object and effect categories) and the discrete relations, Here, the stack action is included into the domain file, with the preconditions and effects given in Fig. 2. The predicates used in planning are given in Fig. 3. We introduced a number of predicates for the tower-building task. `(pickloc ?x)` and `(stackloc ?x)` predicates are set to true if the objects are at pick-up and stacked locations, respectively. `(instack ?x)` predicate is set to true if the effect category is in the following set: {INSERTED, STACKED1, STACKED2}.

Fig. 4 gives sample action descriptions generated from rules 01, 03, and 14, which have effects of TUMBLED1, INSERTED and STACKED1, respectively. We also added predicates that represent the height of the stack ('H') and the number of the objects ('S') in the stack. As the increment operator is not supported in STRIPS notation, a separate action is automatically generated for each level of height and stack increment.

While INSERTED does not affect the height of the stack significantly, STACKED1 and STACKED2 do increase the height according to the mean position changes of objects provided in Table III. Thus, 'H' is increased in rule (01), but not in rule (03). The number of objects is increased with STACKED1 and INSERTED effects, not with the TUMBLED1 effect; thus there is no change in 'S' in TUMBLED1 effect (rule no 14). Finally, in this study we assumed that the category of the tower/stack is determined by the latest included element. For example, if a ROLLABLE object is added to the stack with SOLID or HOLLOW object on top, the next category of the stack is set as ROLLABLE. We plan to relax this constraint in our next work, by learning a rule that predicts the next 'category' of the stack, depending on the categories and relations of the pairs of objects being stacked. In this way, the system should learn that the category of the stack should be kept HOLLOW, if a very small ROLLABLE object is dropped-on a big HOLLOW object.

```
(:action stack ;;; rule no: 01
:parameters (?Below ?Above)
:precondition (and (pickloc ?Above)
(stackloc ?Below) (hollow ?Below) (H0) (S0)
(below-smaller ?Below ?Above) (hollow ?Above) )
:effect
(not (pickloc ?Above)) (not (H0)) (H1) (not (S0)) (S1)
(instack ?Above) (stackloc ?Above)
(not (stackloc ?Below)))
```

```
(:action stack ;;; rule no: 03
:parameters (?Below ?Above)
:precondition (and (pickloc ?Above)
(stackloc ?Below) (hollow ?Below) (S0)
(below-smaller ?Below ?Above) (rollable ?Above) )
:effect
(not (pickloc ?Above)) (not (S0)) (S1)
(instack ?Above) (stackloc ?Above)
(not (stackloc ?Below)))
```

```
(:action stack ;;; rule no: 14
:parameters (?Below ?ABOVE)
:precondition (and (pickloc ?Above)
(stackloc ?Below) (rollable ?Below))
:effect
(not (pickloc ?ABOVE)))
```

Fig. 4. Automatically generated sample actions in PDDL.

```
(define (problem simple-1)
(:domain stack)
(:objects o1 o2 table)
(:init (stackloc table)
(solid table)
(pickloc o1) (pickloc o2)
(hollow o1) (solid o2)
(below-bigger o1 o2)
(H0) (S0))
(:goal (and (S2) (H2))))
```

```
PLAN:
1 (stack table o2)
2 (stack o2 o1)
```

Fig. 5. Left: Sample domain and problem descriptions that include initial world state and the goal. Initial world contains one HOLLOW and one SOLID object, and an empty stack `(S0)`, with 0 height, `(H0)`. The goal is to build a stack of height 2 `(H2)` with 2 objects `(S2)`. Right: The generated plan first stacks the SOLID object onto the empty table, then stacks the HOLLOW object on top of the SOLID.

The goal/problem is defined with STRIPS notation as well. When we define the initial world state, we also introduce a location for tower-building by `(stackloc table)` predicate where table is a SOLID object. Fig. 5 shows a sample world with two objects, that were categorized as HOLLOW and SOLID. The goal is to obtain a predicate with `(H2)`, i.e. a stack of height 2. In order to achieve this goal, the system plans to stack the HOLLOW object on top of the SOLID object. But with the same objects, if the goal is set to have two objects in a stack of height 1, `(S2) (H1)`, then the plan would be different, stacking SOLID object on the bigger HOLLOW one. Note that, if the height of the stack is not important, the tower-building goal can also be defined by using the `(instack o)` predicate.

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.

F. Real World Experiment

In this section, our system is partially validated by real-robot experiments, where the effect category prediction

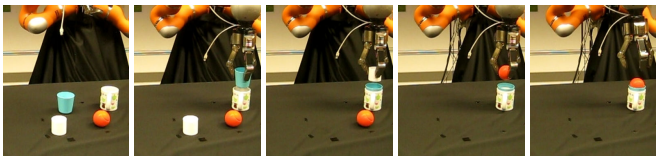


Fig. 6. Tower building experiment. The goal is to build a tower of height 1 with 4 objects. The plan first stacks the HOLLOW objects on top of each other, starting from the bigger ones; and stacks the ball at the end.

(learned in the real world) and the rules (learned in the simulator as described above) are used to build and execute plans in a tower-building task. Given a number of objects, the task of the robot is to build compact towers with all objects included. Thus, the robot needs to plan a sequence of stack actions, and ensure that all objects are stacked with minimal final height of a single tower. Such a task is useful in transporting objects together, to ensure the stability by keeping the stack compact.

Given objects, the robot first finds the object categories using the trained classifiers, and generates the domain description based on object categories. Next, it runs the planner, setting the 'S' predicate to the number of objects, and the 'H' predicate to the minimum number (1) initially. In case no plan is found, the constraint is relaxed, i.e. the goal 'H' predicate is increased by one in a loop, until a plan is found. After the plan is constructed, the robot selects the next objects to stack according to the plan and sequentially executes the stack actions. As the planned effect categories are grounded, the robot can also monitor the plan execution, and check whether the observed effect categories are in accordance with the expected ones; but this feature is not implemented in the current system.

In the first experiment, three cups and one ball is presented to the robot as shown in Fig. 6. The robot first detects that they belong to 'HOLLOW' and 'ROLLABLE' categories, and encodes the category information along with the relative size and height relations in the initial world state as follows:

```
(define (problem tower-real-1)
(:domain stack) (:objects o0 o1 o2 o3 table)
(:init (stackloc table) (solid table) (S0) (H0)
(pickloc o0) (pickloc o1) (pickloc o2) (pickloc o3)
(hollow o0) (hollow o1) (rollable o2) (hollow o3)
;;; width and height relations
(:goal (and (S4) (H1))))
```

Above, the objects correspond to the white-cup, the green-cup, the ball, and the bigger coffee mug, respectively. The robot generated a plan where the HOLLOW objects are stacked on top of each other in decreasing order of width first, and the ball is stacked as the final action. The actions were successfully executed, and a compact and complete tower is built as shown in the snapshots of Fig. 6.

In the second case study, three cups, one ball and a cylindrical shaped salt container are presented to the robot as shown in Fig. 7. The robot first detected that they belong to HOLLOW, ROLLABLE and SOLID. Next, it encoded the category information along with the relative size and height relations in the initial world state as follows:

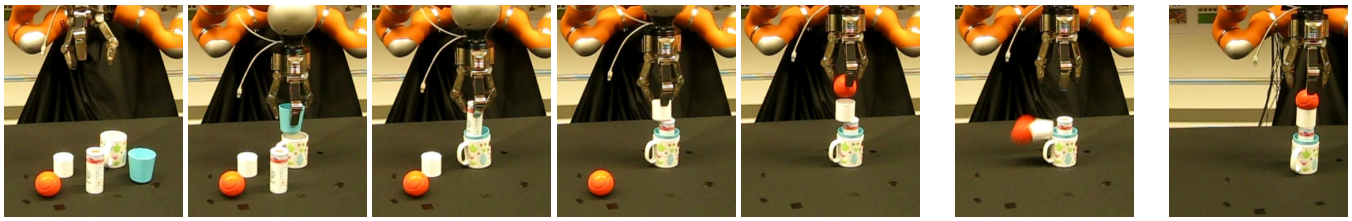


Fig. 7. Tower building experiment. The goal is to build a tower of `(S5)` and `(H2)`. The generated plan first stacks two HOLLOW objects on top of each other, starting from the biggest one; adding salt-container and small cup next, and finally stacking the ball at the end. The execution of the same plan fails in one execution, and succeeds in the other one due to the noise in the perception and the actuation. Please see the robot execution video at http://emreugur.net/icra2015/ for details.

```
(define (problem tower-simple)
(:domain stack) (:objects o0 o1 o2 o3 o4 table)
(:init (stackloc table) (solid table) (H0) (S0)
(rollable o0) (hollow o1) (solid o2) (hollow o3)
(hollow o4) (pickloc o0) (pickloc o1) (pickloc o2)
;;; relations here
(:goal (and (S5) (H1))))
```

A plan was not generated with these constraints, thus the constraints were relaxed by incrementing the target height of the tower: `(:goal (and (S5) (H2)))`. The plan generated for this goal is as follows: the largest two HOLLOW objects and the salt-container are 'inserted' in each other first; then the white-cup is 'stacked', increasing the size; and finally the ball is 'inserted'. This plan was executed two times, leading to successful and unsuccessful results at the end, as shown in the final snapshots of Fig. 7. First of all, the inconsistency between two plan executions were partly due to the inaccuracies in computing the object centers that are used for aligning objects during stacking. One can also notice that the height of the tower is more than `(H2)`, i.e. more than height of 2 objects, because, contrary to the action predictions, the 'inserted' tall salt container and the 'inserted' ball increased the height considerably. A more compact tower could have been obtained by inserting all cups into each other and inserting salt-container into the inserted cups. Also, note that while object category classification is correct in all instances, the rules that were learned in the simulator are not always valid. For example, the action with the 'stacked' effect is not reliable, i.e. the objects tumble more easily in the real world. Re-learning in the real world while building the towers is necessary to handle these situations.

IV. Conclusion

In this study, we developed a system that forms symbols and operators in the continuous sensorimotor experience of the robot through self-exploration. These formed structures were used to generate symbolic plans in the robot in a sample tower building task. While study provides a proof-of-the-concept realization of the proposed system, it only uses one action, namely stacking, in plan generation and execution. Our system should be verified with a richer set of action repertoire where principles that regulate the unsupervised effect categorization process (such as the number of clusters or the feature space that is being clustered) should be investigated in detail. We also introduced a number of discrete features such as smaller-than, and a number of task-related predicates such as pick-loc and stack-loc; which should be discovered by the system autonomously in the future.

The most immediate extension of this research is to incorporate probabilistic techniques in discovering symbols, learning operators and generating plans. Particularly, we plan to construct probabilistic rules from the robot's noisy interactions based on category prediction confidences, and use planners that can deal with incomplete and noisy information [12]. During our experiments, we observed that, in addition to noise in sensing and perception, failures in plan execution were partially due to the simple encoding of the currently used motion primitives and the object perception that only relies on vision. Therefore, our system should incorporate vision and force guided closed-loop motion representation frameworks.

References

[1] V. Klingspor, K. Morik, and A. D. Rieger, "Learning concepts from sensor data of a mobile robot," *Machine Learning*, vol. 23, no. 2-3, pp. 305–332, 1996.

[2] F. Wörgötter, A. Agostini, N. Krüger, N. Shylo, and B. Porr, "Cognitive agents: A procedural perspective relying on the predictability of Object-Action-Complexes OACs," *Robotics and Autonomous Systems*, vol. 57, no. 4, pp. 420–432, Apr. 2009.

[3] K. Mourao, R. P. Petrick, and M. Steedman, "Using kernel perceptrons to learn action effects for planning," in *CogSys*, 2008, pp. 45–50.

[4] R. Sun, "Symbol grounding: A new look at an old idea," *Philosophical Psychology*, vol. 13, no. 149–172, 2000.

[5] 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.

[6] 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.

[7] J. Kulick, M. Toussaint, T. Lang, and M. Lopes, "Active learning for teaching a robot grounded relational symbols," in *Proc. 23rd Int. Joint. Conf. AI*. AAAI Press, 2013, pp. 1451–1457.

[8] 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.

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

[10] E. Ugur, Y. Nagai, E. Sahin, and E. Oztop, "Staged development of robot skills: Behavior formation, affordance learning and imitation with motionese," *IEEE Transactions on Autonomous Mental Development*, 2015, submitted.

[11] C.-C. Chang and C.-J. Lin, "Libsvm : a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 27, pp. 1–27, 2011.

[12] R. P. A. Petrick and F. Bacchus, "Extending the knowledge-based approach to planning with incomplete information and sensing," in *ICAPS*, 2004, pp. 2–11.