2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Robot Learning and Use of Affordances in Goal-directed Tasks

Chang Wang[1], Koen V. Hindriks[1] and Robert Babuska[2]

Abstract— **An affordance is a relation between an object, an action, and the effect of that action in a given environmental context. One key benefit of the concept of affordance is that it provides information about the consequence of an action which can be stored and reused in a range of tasks that a robot needs to learn and perform. In this paper, we address the challenge of the on-line learning and use of affordances simultaneously while performing goal-directed tasks. This requires efficient on-line performance to ensure the robot is able to achieve its goal fast. By providing conceptual knowledge of action possibilities and desired effects, we show that a humanoid robot NAO can learn and use affordances in two different task settings. We demonstrate the effectiveness of this approach by integrating affordances into an Extended Classifier System for learning general rules in a reinforcement learning framework. Our experimental results show significant speedups in learning how a robot solves a given task.**

I. INTRODUCTION

Many real world robotic tasks, like navigation or object manipulation, are dynamic and require on-line performance. Moreover, a fully preprogrammed approach is not sufficient to handle the underlying uncertainties of environments. One solution is that robots learn autonomously through observations and embodied interactions with environments. Specifically, in a goal-directed task where a robot interacts with objects, it learns to optimize its policy and select actions with a higher chance of success. If the task or the objects are changed, the previously learned policy will probably no longer be optimal. Relearning a new policy from scratch is not effective. In order to efficiently construct a new optimal policy, it is useful to extract information from the previously learned tasks. The notion of affordance [4] provides robots with information whether an object affords an action or not [15], and can be modeled as a relation between an objects, an action and an effect [10]. This information is useful for action selection in on-line learning tasks in which repetitive trials are usually required for learning an optimal policy. In this paper, our main goal and contribution is to investigate and propose a framework that combines on-line learning of affordances and the use of affordances at the same time to improve the robot's learning performance in a goal-directed task.

Integrating on-line learning of affordances and the use of that learned knowledge is a challenge. First, affordance learning usually requires several steps of training to build the relations between the perception space of objects and the perception space of effects, both of which involve machine learning techniques to classify objects and effects [19]. Moreover, many affordance learning approaches are based on a developmental framework [1] where a robot first learns affordances in a goal-free self-exploration stage, whereafter it uses the affordances for goal-directed behaviors [2][19]. These approaches thus separate affordance learning and use, so that there is no on-line learning of affordance in the latter phase. As a result, they have difficulty handling environmental changes. For example, when the robot has learned that a ball is rollable during the training process, it might end up repetitively trying to kick an unmovable object which looks similar to the ball. In contrast to these approaches, our aim is to provide an approach that integrates the learning and the use of affordances which allows learning while the robot performs a task, even in a dynamic environment.

Due to the complexity of real world environments and the limitations of robot platforms, it is a big challenge to establish affordances for a multitude of robotic behaviors in various environments [11]. In the literature, some affordances are predefined based on human-predicted effects such as rollability [3], liftability [12] and traversability [20]. For example, a ball will probably roll by a pushing action and a box will not. A wide range of machine learning techniques have been proposed for learning affordances. For example, Support Vector Machines were trained in an exploration phase to predict the possibility of traversing when the robot faces different shaped objects [20]. Other methods include Decision Trees [16], Bayesian Networks [10], Self-Organizing Maps [14] and Reinforcement Learning [13]. However, affordances were not learned on-line [10][16], or reused in other tasks [13][14][25] due to that only one specific action was given to the robot to learn affordances. Furthermore, most of the existing methods have focused on the discussion of how a robot perceives affordances rather than how a robot makes use of affordances to improve its performance in tasks.

The main contribution of this paper is the proposal of an architecture that integrates simultaneously on-line learning and the use of affordances in goal-directed tasks. Affordances are stored as interpretable triples in a table that can be updated and reused in a set of tasks. More specifically, affordances are acquired automatically during on-line task learning. But, while being learned, they are *also* used to speed up the task learning. In our approach, affordance learning interacts with a task learning system, using an XCS classifier system [24], within a reinforcement learning [18]

[1]C. Wang and K.V. Hindriks are with the Interactive Intelligence Group, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, 2628 CD Delft, The Netherlands.

[2] R. Babuska is with the Delft Center for Systems and Control, Faculty of Mechanical Engineering, Delft University of Technology, 2628 CD Delft, The Netherlands.

All authors are with the TU Delft Robotics Institute, robotics.tudelft.nl. {c.wang-2, k.v.hindriks,r.babuska}@tudelft.nl

framework. In addition, we pay special attention to the on-line use of affordances.

The paper is organized as follows: Section II describes the interaction between affordances and task learning. Section III introduces the task environments that we use in our experiments. Section IV concludes the paper and outlines our plans for future work.

II. Simultaneous Learning and Use of Affordances During Task Learning

Similarly to [10], [15], we define an affordance as a triple:

$$(Object, Action, Effect) \tag{1}$$

where *Object* refers to the entity that can be interacted with, e.g., a box or a door knob; *Action* refers to a behavior or repertoire of motor skills that can be used to interact with the object, e.g., pushing and turning; and, *Effect* refers to the result of performing the action on the object, e.g., the box has been moved or the door is opened. We note that affordances provide general information about the effects of actions on objects and this knowledge is independent of the task at hand.

A. Architecture

The overall approach to the learning and use of affordances while learning how to perform a goal-directed task is based on an architecture consisting of three components: an affordance learning component, a common task learning component based on reinforcement learning, and a component that feeds learned affordances into the action selection mechanism of the task learning component. Fig. 1 illustrates the architecture we propose.

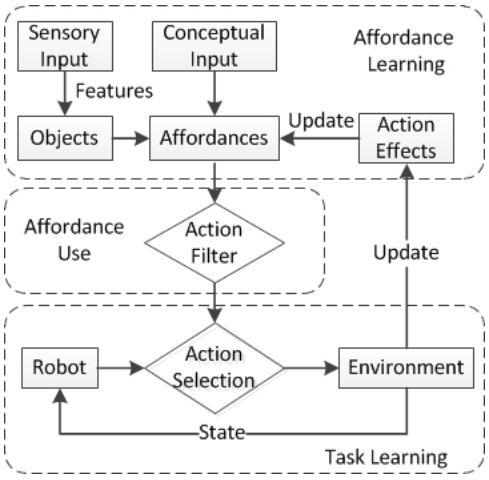


Fig. 1. An architecture for affordance learning and use during task learning.

In the affordance learning component, the robot perceives the environment via its sensors. It extracts features and forms a representation of the environment. For example, a camera provides visual features of an object such as the color or shape and a depth sensor provides the distance between the robot and the object. The robot is provided with a set of preprogrammed behaviors to learn affordances and perform tasks. Additionally, the conceptual input provides the robot with the knowledge on how the effects of these actions are detected, e.g., a notion of distance and its change. By providing the affordance learning component with a priori conceptual knowledge about general types of affordances, actual affordances can be learned in association with particular objects and robot actions. In other words, the affordances are updated on-line during the task learning, by collecting data of the encountered object, the action performed and the resulting effect. For example, a robot can learn the movability of objects by observing their displacement after a pushing action. In summary, with the sensory input, given actions, and the conceptual knowledge, the robot is able to learn actual affordances during its embodied interaction with the environment.

In the affordance use component of Fig. 1, the learned affordances filter out those actions which are likely to cause undesired effects under the current task goal. As the general problem of relating such effects with goals is beyond the scope of this paper, we use a simple approach where we assume that when an action has no effect on the object, performing the action is considered to be undesired. For example, the robot pushes an unmovable box and learns that it cannot move the box, thereafter it avoids pushing this box.

Finally, in the task learning component of Fig. 1, the task is learned under a goal in a reinforcement learning framework, which as usual consists of a set of states and actions, as well as rules describing state transitions and the associated immediate rewards.

The state vector contains sufficient information about the object for the current task, the set of actions is the same set of preprogrammed behaviors as for learning affordances, and the reward is derived from the task goal. The task learning system makes use of the affordances for action selection, avoiding inefficient actions and thus resulting in a higher chance of selecting an action sequence to achieve the task goal. In other words, the affordances bias the action selection and speed up the task learning process.

Even though affordance and task learning are coupled in our approach, the affordances learned are independent from task learning and can be reused in other tasks.

B. Affordance Learning

We have defined affordances as triples above, see (1). We now discuss in more detail each of the three components of an affordance.

1) Perception of objects: The robot perceives the environment and extracts a set of features $\{f_i\}_{i=1}^{N}$ from its raw sensory input. In this paper, objects are classified based on features such as color and size. Then, the object is denoted by o which refers to the set of attributes. We also assume that the sensors can be used to extract the environmental state of the object which might be changing with time, denoted as s_o, e.g., its location in the world space.

2) Robot actions: We provide the robot with a set of preprogrammed behaviors $\{a_j\}_{j=1}^{M}$ that can be used to perform a given task. For example, a robot may encounter obstacles

Algorithm 1 On-line learning of affordances during a task

1: Initialize the affordance table [T] empty or load it from a file.
2: Repeat until the end of the task
3: Observe the current object o and its state s_o.
4: Apply action a.
5: Observe a new state s'_o.
6: Compute the effect e of a by using (2).
7: **if** (o, a) matches no item in [T] **then**
8: Add (o, a, e) to [T].
9: **else**
10: Replace (o_k, a_k, e_k) by (o, a, e) if $o = o_k$ and $a = a_k$.
11: **end if**
12: Go to Line 2.

while navigating through a corridor. In our affordance-based approach, if an obstacle is detected in front of the robot, it can choose to remove the obstacle, e.g., push it aside, lift it or it can choose to step over it.

3) Perception of effect: After performing an action a on the object o, the robot perceives a new state of the object s'_o. By comparing s'_o and s_o, the effect e can be obtained as follows:

$$e = m(s'_o, s_o) \tag{2}$$

where m is a suitable measure. For example, a box is pushed and its location change is measured. If a robot tries to drop an object into a rubbish bin, the effect can be a binary state description that the object is in the rubbish bin or not.

4) Update of affordances: The knowledge of affordances is represented as an affordance table [T], consisting of triplets:

$$(o, a, e) \tag{3}$$

in which o is the object, a is the action and e is the effect.

Algorithm 1 updates [T] during a task. Initially, [T] is empty or loaded from a file which contains previously learned affordances. Various mechanisms can be used to update the affordances. In this paper, we use a short term memory with a simple forgetting mechanism. That is, the robot obtains (o, a, e) and searches (o, a) in [T]. If no match is found, (o, a, e) is added to [T]. Otherwise, assume $o = o_k$ and $a = a_k$, then (o_k, a_k, e_k) is replaced by (o, a, e). In this way, the robot is able to handle dynamic situations in which the action effect on the same object changes. Although the current implementation is a simple affordance table, it allows further generalization with additional techniques.

C. Task Learning System

We demonstrate the effectiveness of the architecture by using an Extended Classifier System (XCS) for learning general rules in a reinforcement learning context.

XCS classifier system [24] is a rule-based system which solves reinforcement learning problems [18]. It can be regarded as a generalization of tabular Q-learning [22] by using a Genetic Algorithm (GA) [5] to aggregate equivalent states in the Q-table [6]. In XCS, the GA produces rules which are used by Q-learning to update the prediction of reward. Then, based on the error of prediction, the rules are evaluated by their fitness values and updated by the GA. Besides, while tabular Q-learning updates a single state-action pair, XCS updates multiple state-action pairs. For physical robot control, XCS does not require careful tuning of parameters to achieve satisfactory behavior [17]. So we choose XCS for on-line robot learning tasks.

The knowledge of XCS is represented as a set of rules (called classifiers in the LCS literature [7]). The rules can use real numbers, symbols or the classical ternary representation. In this paper, we choose the ternary representation {0,1,#} to encode the conditions and binary strings to encode the actions. The hash symbol # can be either 0 or 1 which allows generalization and GA operations on the rule conditions with the same length.

A rule maps a condition and an action to a prediction, with an associated fitness as follows:

$$(condition, action) \rightarrow prediction : fitness \tag{4}$$

For example, the rule

$$(0\#0\#11\#11, 001) \rightarrow 1000 : 0.59 \tag{5}$$

means if the current state string s meets the condition $0\#0\#11\#11$ and if action 001 is taken, then a reward of 1000 is predicted. This rule has a fitness of 0.59.

The mapping in equation (5) is a value function, so the rules are value function fragments, and XCS generalizes over its value function using GA techniques [21].

In this paper, the current state of the system (see Fig. 1) only considers the object, represented as:

$$s = (o, s_o) \tag{6}$$

which ignores the information about the robot itself, e.g., its distance to the object and its joint angle values. We assume that the objects are within the robot's reach and we note that the state of the robot is not included in the tasks considered in this paper. However, in general, it may be included.

Assume [P] is the current set of rules of XCS. If s is obtained from the environment by the robot sensors, then a subset of [P] forms a match set [M] whose rule conditions match s. [M] can be further decomposed as a union of subsets:

$$[\mathrm{M}] = \bigcup_{i=1}^{M} [\mathrm{M}]_{a_i} \tag{7}$$

where $[\mathrm{M}]_{a_i}$ are the rules which advocate action a_i. Then, a prediction of action a_i is calculated as

$$P(a_i) = \frac{\Sigma_{l \in [\mathrm{M}]_{a_i}} p_l F_l}{\Sigma_{l \in [\mathrm{M}]_{a_i}} F_l} \tag{8}$$

in which p_l is rule l's prediction of reward and F_l is rule l' fitness, based inversely on the error ϵ_l in the prediction of p_l. Based on equation (8), an action a_j is selected by

$$a_j = \arg\max_i P(a_i) \tag{9}$$

where a_j is the action with maximal prediction. After a_j is applied and reward r is obtained, all the rules in $[\mathrm{M}]_{a_j}$ are updated by the *Credit Assignment Algorithm* [24], which uses a version of Q-learning update to distribute r. For more information on updating p_j, ϵ_j and F_j, we refer the reader to [21].

D. Affordance Use

Traditionally, the task learning system selects an action according to a criterion, e.g., random action selection, greedy action selection or mixed. Some actions are efficient in the learning task while some are not. Take a navigation task for example, the robot would prefer pushing a movable obstacle to the side if this helps arrive at its destination faster than by avoiding the obstacle, but trying to push the same unmovable obstacle for several times is not desired. In the task learning system, however, there is no guarantee that this will not happen.

We show that this problem can be solved with the aid of affordances. Under a specific task goal, some effects are desired while some are not. In this paper, if o matches o_k in [T], the related action a_k is filtered out from the candidate actions if its effect e_k satisfies:

$$e_k \in \mathscr{E} \tag{10}$$

where $\mathscr{E}$ is the set of predefined undesired effects. For example, "unmoved" is not desired in a task where the goal is to move objects to a location, while "moved" is not desired in a task where the robot is not allowed to move anything when navigating to the destination.

In this way, affordances influence action selection of the task learning system. Algorithm 2 shows how the affordances are used by the XCS classifier system.

Algorithm 2 Use affordances with XCS in a task

1: Initialize the affordance table [T] and the population of XCS rules [P] empty or load them from files.
2: Repeat until the end of the task
3: Observe the current system state $s = (o, s_o)$.
4: Form [M] $\subset$ [P] whose rule conditions match s.
5: Calculate the prediction for each action using (8).
6: **if** o matches object o_k of (o_k, a_k, e_k) in [T] **then**
7: Filter a_k by its effect e_k using (10).
8: **end if**
9: Select an action a_j by using (9).
10: Apply a_j.
11: Observe the new system state $s' = (o, s'_o)$.
12: Update [T] by Algorithm 1 (Line 6 to Line 11).
13: Receive a reward r from the environment.
14: Form the action set $[\mathrm{M}]_{a_j} \subset [\mathrm{M}]$ which advocated a_j.
15: Call the *Credit Assignment Algorithm* to update $[\mathrm{M}]_{a_j}$.
16: Go to Line 2.

At the first time step, the system initializes [T] and [P], both of which can be empty or loaded from files. Line 2 starts a loop that the robot selects an action in explore or exploit episode, alternating until the end of the task. In case of an endless loop, each episode ends anyway after n_{step} steps. Line 4 and 5 are the standard XCS way of forming the match set [M] and prediction of actions. Before selecting an action in Line 9, Line 6 checks [T] first. If o matches an object o_k of (o_k, a_k, e_k) in [T], equation (10) filters a_k by its effect e_k. After the robot performs the action a_j, it observes the new system state s' and updates [T] by algorithm 1. At last, the action set $[\mathrm{M}]_{a_j} \subset [\mathrm{M}]$ which advocated a_j is updated by the *Credit Assignment Algorithm* [6].

III. Experiment: Learn Movability in Goal-directed Tasks

In this paper, we investigate the movability of objects with different weights and sizes. A humanoid robot uses its whole body motion to push the object in front of it. It observes the effect, which is the location change of the object before and after the action. With these affordances, the robot is expected to operate more effectively and avoid making mistakes repeatedly in a task, e.g., pushing the same unmovable box even though it has failed for several times.

Sequential goal-directed tasks are designed to test the proposed model on a humanoid robot NAO. We assume that the robot is able to detect the current state of the environment and of itself by extracting features from its camera and sonars. In our setup, the NAO has a repertoire of preprogrammed behaviors to interact with the environment. We use XCSlib [8] for task learning.

A. Environment and tasks

Given a map and a landmark in the world space, the NAO can localize itself and guarantee its safety on a flat table (see Fig. 2). A marker is used on each object to measure the relative location and distance to the NAO.

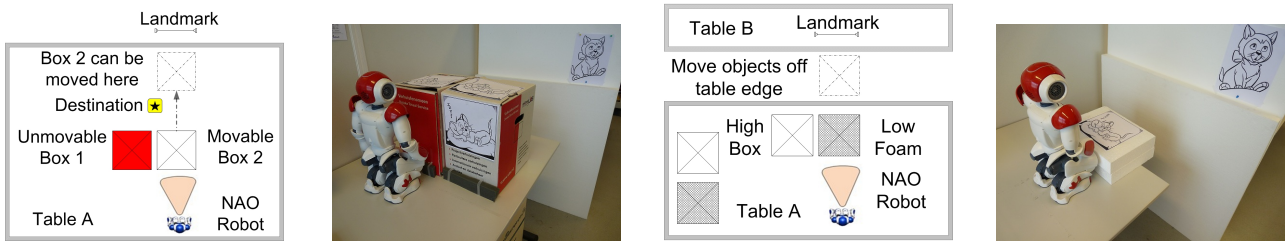
In the navigation task, two boxes block the NAO's way and they are movable and not movable respectively, see Fig. 2(a) and Fig. 2(b). The goal is to reach the destination as quickly as possible. After several trials, the locations of the boxes are exchanged to show that the NAO can reuse the learned movability in a different setting.

In the stacking task, the NAO stands on the table edge and chooses different pushing poses for a high box or a piece of low foam, see Figures 2(c) and 2(d). The goal is to push them off the table to make a stack as high as the table.

B. Sensory Input

The forehead camera is used as the main sensory input (640×480 resolution) and the sonars on its chest confirm there is an object in front of the robot. The NAO localizes itself by matching SIFT [9] features of the landmark with known 3D coordinate (x_l, y_l, z_l) in the world space. This provides the NAO its camera location (x_r, y_r, z_r) in the world space [23]. When the marker on the object is detected, the NAO calculates its relative location to the object as $(x_{r2o}, y_{r2o}, z_{r2o})$. The object's location (x_o, y_o, z_o) is then obtained, which is discretized as a 4-bit binary string S_{xyz} to represent s_o. Meanwhile, the height of the object h_o is obtained by:

$$h_o = h_{r2t} - |z_{r2o}| \tag{11}$$



(a) A navigation task. (b) NAO faces two boxes. (c) A stacking task. (d) NAO faces a piece of foam.

Fig. 2. A humanoid robot NAO learns movability in a navigation task and a stacking task.

where h_{r2t} is assumed known as the height of the NAO's camera in its normal standing pose and $|z_{r2o}|$ is the approximate vertical distance from the camera to the object. Then, h_o is represented by o_{height} where 1 means the object is high and 0 means low. The colors of the boxes are represented by a 3-bit binary string o_{color}. In total, s in equation (12) is a 8-bit string:

$$s = \left(o, s_{xyz}\right) \tag{12}$$

in which

$$o = \left(o_{height}, o_{color}\right) \tag{13}$$

C. Actions

We provide the NAO with eight actions represented as a 3-bit binary string in Table I. In the navigation task, the first four actions are used. The NAO can move a distance of d_{walk} left, right, ahead, or push when walking ahead. In the stacking task, the NAO stands on the table edge and chooses the latter four actions in Table I. To simplify the process of finding and taking an object to the table edge, the NAO chooses an object by requesting the human operator with an "text-to-speech" command. Then, the NAO confirms there is an object in front and tries to push it in two different poses, standing or squatting.

TABLE I
ACTIONS OF THE NAO FOR THE TASKS

Binary ID	Actions
000	walk (d_{walk}) left
001	walk (d_{walk}) ahead
011	walk (d_{walk}) right
010	push (d_{walk}) ahead
100	push (stand)
101	push (squat)
110	choose (object 1)
111	choose (object 2)

D. Movability

As defined in equation (2) and (3), movability is learned automatically in a task whenever the robot has effective contact with the objects, e.g., pushing them in various ways. In our case, pushing the object o with the action a would make it move a distance of d_j, which is thresholded to be 1 or 0, meaning "moved" or "unmoved" respectively.

E. Action Filter

The action filter in equation (10) blocks those actions which are predicted to have the "unmoved" effect on the object. For example, pushing an unmovable object or pushing a movable low object in a wrong pose.

F. Reward Function

After the action a_j at the previous time step, the NAO gets an immediate reward of r in the current time step. In the navigation task, if its current distance to the destination d is smaller than a threshold d_{dst}, a final reward of 1000 is given. If its location change Δd is less than d_{move}, it is punished by -100. This is described by:

$$r = \begin{cases} 1000 & \text{if} \quad d < d_{dst} \\ -100 & \text{if} \quad \Delta d < d_{move} \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

In the stacking task, if the distance from the NAO's camera to the object approximately equals h_{r2t}, the distance from the NAO's camera to the table, a final reward of 1000 is given. Whenever it asks for an object, it gets a punishment of -100. Otherwise, $r = 0$. This is described in equation (15):

$$r = \begin{cases} 1000 & \text{if} \quad |h_{r2f} - h_{r2t}| < h_\epsilon \\ -100 & \text{if} \quad a_j = \text{choose (object)} \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

When each explore/exploit trial terminates, the rewards obtained in this trial are summed up to be the accumulated *payoff*:

$$payoff = \Sigma\, r \tag{16}$$

G. Experimental setting

In both tasks, an experiment consists of alternated explore and exploit trials. They use the same parameter setting of XCS. For more details, please refer to [21]. The number of steps in one trial is limited to the maximum of $n_{step} = 10$.

During the first 10 explore/exploit trials in the navigation task, the unmovable box is put on the left side of the movable one (see Fig. 2(b)). Then, the boxes are exchanged and the remaining 10 trials are performed. In this task, the table area is $0.75\,\text{m} \times 0.6\,\text{m}$, and $d_{walk} = 0.15\,\text{m}$. The NAO can start from anywhere on one side of the area aiming for the destination which is $0.4\,\text{m}$ away on the other side of the area.

In the stacking task, the camera height of the NAO in its normal standing pose is known as $h_{r2t} = 0.55\,\text{m}$.

TABLE II

XCS RULES LEARNED IN THE NAVIGATION TASK.

ID	Condition	Act.	Pred.	Error	Fit.	Size	E.	N
12	1#00#010	000	-33.6	29.7	0.29	2.5	2	3
14	1#1#####	000	24.3	71.7	0.32	5.6	11	5
30	#1100#00	000	-93.0	1.0	0.37	3.0	2	3
137	1#1#####	000	109.9	24.8	0.15	7.3	3	1
23	1000####	001	12.5	54.8	0.31	3.7	3	3
24	10000###	001	-44.0	25.5	0.02	3.0	0	1
25	1011####	001	10.0	1.0	0.01	1.0	0	1
36	#11#0010	001	-93.0	1.0	0.37	1.0	1	1
60	11#1####	001	10.0	1.0	0.01	1.0	0	1
2	1####000	010	138.1	137.9	0.15	6.9	10	3
7	###01100	010	850.0	75.0	0.37	1.0	2	2
9	##0#10##	010	241.7	154.0	0.68	2.6	5	2
19	10#0#110	010	1000	1.0	0.59	3.3	4	3
39	1111####	010	-93.0	1.0	0.37	1.0	1	1
45	1###001#	010	493.1	138.4	0.23	3.6	5	2
94	1###00#0	010	456.0	93.0	0.33	7.4	6	2
106	#00001#0	010	917.3	89.6	0.07	6.3	4	1
146	1#####00	010	672.1	163.0	0.14	8.7	3	1
5	###01000	011	-94.0	0.5	0.30	2.5	2	2
21	100#1010	011	-33.6	29.7	0.17	2.5	2	3
26	#1#000#0	011	-111.9	9.5	0.37	1.5	2	2

H. Results

We compare the results of the standard XCS and the affordance-based XCS in the navigation task, and we show the affordances learned in both the navigation and the stacking task.

1) XCS Rules: Table II lists a typical population of XCS rules learned in one experiment. The rules with larger ID were generated after the rules with smaller ID.

The meaning of the rules is according to equations (12) and (13). Take rule 19 for example, when facing the object "10#0" (high and probably a white box) which is in the area "#110" (the middle part of the table, either left or right), if the NAO takes the action "010" (push 0.15 m ahead), the predicted reward will be 1000 (goal achieved), and the error of this prediction is 1.0. This rule has a fitness of 0.59, and the average action set size is 3.3. It has been activated for 4 times and there are 3 such rules with the same condition and action.

In the first 10 trials, if the NAO faced the object "1000" in the area "1000", it was most likely to choose rule 9, 7 and 19 to push the movable box. In the second 10 trials, if the NAO faced "1111" also in the area "1000", it was inclined to choose rule 14 and 137 to move 2 steps left to avoid pushing the unmovable box.

2) Affordance-Based XCS versus Standard XCS: As mentioned in Section III-G, odd trials were always exploratory while even trials were exploiting. The average payoff and the average number of steps in each trial are shown in Figures 3 and 4. The result was averaged over 5 experiments.

In the first 10 trials, both affordance-based XCS and standard XCS solved the problem since the 4th trial. This is because it was not difficult for the NAO to find the two-step solution to push the movable box. However, the differences became obvious in the next 10 trials. Affordance-based XCS succeeded in finding the way to the destination while the standard XCS failed within 10 trials. Overly general rules

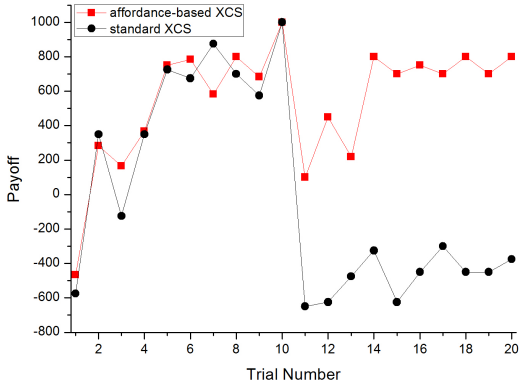


Fig. 3. Average payoff in the navigation task.

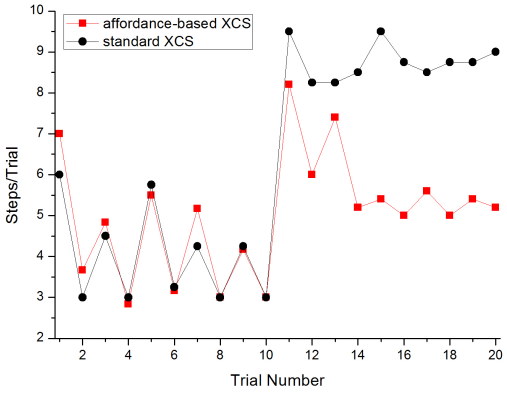


Fig. 4. Average number of steps in the navigation task.

were learned in the first 10 trials which suggested the NAO to push the object to the old place as before. It would take some time for XCS to recover from these incorrect rules. But the affordance-based approach did not suffer from this problem, because it filtered undesired action effects in every step of action selection. Therefore, it had a better chance to achieve the optimal performance than the traditional XCS.

3) Learned Movability: Table III shows the learned movability. Similar to the XCS rules, the triplets with larger ID were generated later. Actually, we can find the relations between the triplets in Table III and the rules in Table II. For example, triplet 1 in Table III is related to rule 39 in Table II. They suggest that action "010" did not move the object "1111" and a reward of -93 is predicted.

In the navigation task, Table III filtered unwanted actions by triplets 1, 2 and 4 which means that the NAO pushed the unmovable box or part of it with no effect. In contrast, triplet 3 means that the NAO pushed the movable box ahead. As a result, the NAO avoided the unmovable box and pushed the movable one to arrive at the destination.

In the stacking task, the NAO first failed to move the low object in a normal standing pose, as illustrated by triplet 5.

TABLE III

MOVABILITY LEARNED IN THE NAVIGATION AND THE STACKING TASK.

ID	Object	Action	Effect
1	1100	010	0
2	1111	010	0
3	1000	010	1
4	1011	010	0
5	0000	100	0
6	0000	101	1
7	1000	100	1

The NAO tried to squat first and then push the low object to move it. Finally, the NAO succeeded to learn the policy to choose the high box rather than the low foam due to the reinforcement.

IV. CONCLUSIONS

In this paper, we proposed an architecture to learn and use affordances on-line in goal-directed tasks. We have integrated task learning, affordance learning and affordance use in a general reinforcement framework, in which affordance information can be obtained automatically during on-line task learning. Our approach has proved to speed up a navigation task and is robust to environmental changes while a standard action selection methods failed to solve the problem within a limited number of steps. In the future, we will further investigate this architecture to learn and use affordances to improve the robot's performance in more complex tasks, without discretizing state and action representations.

REFERENCES

[1] M. Asada, K. F. MacDorman, H. Ishiguro, and Y. Kuniyoshi. Cognitive developmental robotics as a new paradigm for the design of humanoid robots. *Robotics and Autonomous Systems*, 37(2):185–193, 2001.

[2] M. R. Dogar, M. Cakmak, E. Ugur, and E. Sahin. From primitive behaviors to goal-directed behavior using affordances. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems IROS 2007*, pages 729–734, 2007.

[3] P. Fitzpatrick, G. Metta, L. Natale, S. Rao, and G. Sandini. Learning about objects through action - initial steps towards artificial cognition. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 3, pages 3140 – 3145 vol.3, Sept. 2003.

[4] J. J. Gibson. *The ecological approach to visual perception*. Houghton Mifflin, 1979.

[5] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. A Bradford Book, Apr. 1992.

[6] T. Kovacs. *Strength or Accuracy: Credit Assignment in Learning Classifier Systems*. Springer, London, UK, 2004.

[7] P. L. Lanzi. Learning classifier systems: then and now. *Evolutionary Intelligence*, 1:63–82, 2008.

[8] P. L. Lanzi and D. Loiacono. XCSlib: The XCS classifier system library, 2009.

[9] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

[10] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor. Learning object affordances: From sensory–motor coordination to imitation. *IEEE Transactions on Robotics*, 24(1):15 –26, Feb. 2008.

[11] R. Murphy. Case studies of applying Gibson's ecological approach to mobile robots. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 29(1):105–111, 1999.

[12] L. Paletta, G. Fritz, F. Kintzler, J. Irran, and G. Dorffner. Learning to perceive affordances in a framework of developmental embodied cognition. In *Proceedings of IEEE 6th International Conference on Development and Learning*, pages 110 –115, July 2007.

[13] L. Paletta, G. Fritz, F. Kintzler, J. Irran, and G. Dorffner. Perception and developmental learning of affordances in autonomous robots. *KI 2007: Advances in Artificial Intelligence*, pages 235–250, 2007.

[14] B. Ridge, D. Skocaj, and A. Leonardis. Self-supervised cross-modal online learning of basic object affordances for developmental robotic systems. In *Proceedings of 2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5047 –5054, May 2010.

[15] E. Sahin, M. Cakmak, M. R. Dogar, E. Ugur, and G. Ucoluk. To afford or not to afford: A new formalization of affordances toward affordance-based robot control. *Adaptive Behavior*, 15(4):447–472, 2007.

[16] J. Sinapov and A. Stoytchev. Learning and generalization of behavior-grounded tool affordances. In *Proceedings of the 6th International Conference on Development and Learning*, pages 19 –24,July 2007.

[17] M. Studley and L. Bull. X-TCS: accuracy-based learning classifier system robotics. In *The 2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2099–2106, Sept. 2005.

[18] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[19] E. Ugur, E. Oztop, and E. Sahin. Goal emulation and planning in perceptual space using learned affordances. *Robotics and Autonomous Systems*, 59(7–8):580–595, 2011.

[20] E. Ugur and E. Sahin. Traversability: A case study for learning and perceiving affordances in robots. *Adaptive Behavior*, 18(3-4):258–284, 2010.

[21] C. Wang, P. Wiggers, K. Hindriks, and C. Jonker. Learning classifier system on a humanoid NAO robot in dynamic environments. In *12th International Conference on Control, Automation, Robotics and Vision*, pages 94–99, 2012.

[22] C. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[23] C. Wei, J. Xu, C. Wang, P. Wiggers, and K. V. Hindriks. An approach to navigation for the humanoid robot NAO in domestic environments. In *14th Towards Autonomous Robotic Systems (TAROS-13)*, Oxford, U.K., In press 2013. Springer Berlin Heidelberg.

[24] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, June 1995.

[25] T. Yamashiro and K. Ito. Comparative study of affordance-based navigation and model-based navigation: Experimental analysis of learning ability of mobile robot that taps objects with a stick for navigation. In *2011 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 372–377, Dec. 2011.