Use of Affordances for Efficient Robot Learning

Wang, Chang

DOI
10.4233/uuid:ad3f23f8-5bb7-47d1-a42b-bd1043fed661

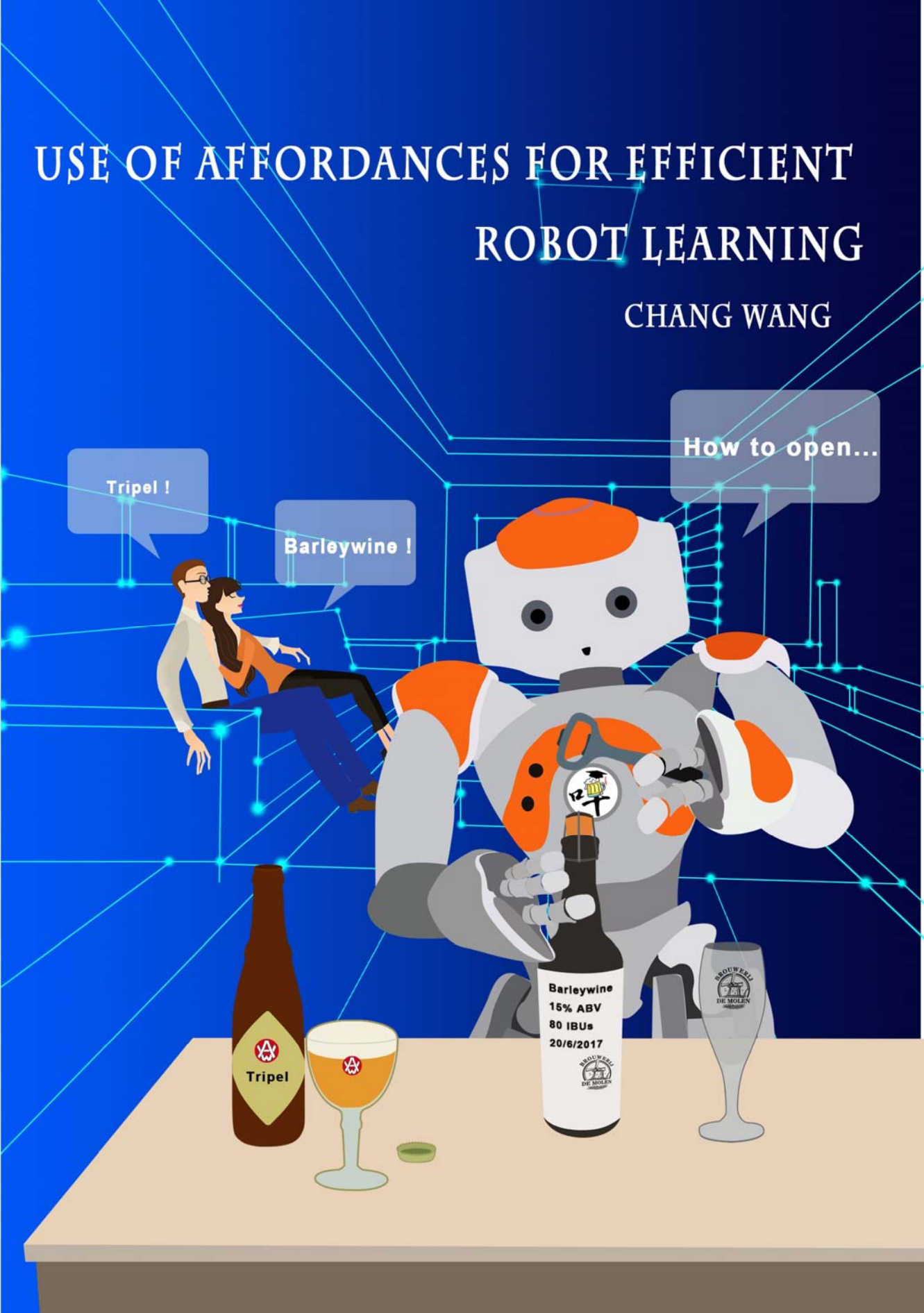
Publication date
2017

Document Version
Final published version

Citation (APA)
Wang, C. (2017). *Use of Affordances for Efficient Robot Learning*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:ad3f23f8-5bb7-47d1-a42b-bd1043fed661

USE OF AFFORDANCES FOR EFFICIENT ROBOT LEARNING

CHANG WANG



Use of Affordances for Efficient Robot Learning

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op dinsdag 20 juni 2017 om 15:00 uur

door

Chang WANG

Master of Science in Applied Mathematics, NUDT, China
geboren te Wuhan, Hubei, China

This dissertation has been approved by the:
promotor: Prof. dr. R. Babuška
copromotor: Dr. K.V. Hindriks

Composition of the doctoral committee:

Rector Magnificus,	chairman
Prof. dr. R. Babuška	Delft University of Technology
Dr. K. V. Hindriks	Delft University of Technology

Independent members:

Prof. dr. C. M. Jonker	Delft University of Technology
Prof. dr. P. P. Jonker	Delft University of Technology
Prof. dr. V. Evers	University of Twente
Prof. dr. M. A. Neerincx	TNO
Prof. dr. A. Nowé	Vrije Universiteit Brussel

SIKS Dissertation Series No. 2017-24.
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

This work is supported by the China Scholarship Council (CSC).

Published and distributed by: Chang Wang.
E-mail: c.wang.tud@outlook.com

ISBN 978-94-6186-814-5

Keywords: Robot Learning, Affordance, Reinforcement Learning, Developmental Robotics.

Printed by Proefschriftmaken, the Netherlands.

Cover artwork by Yeqing Zhou.

Acknowledgements

First and foremost, I thank my supervisor Professor Robert Babuška, who has endless enthusiasm and keeps inspiring me with new ideas. With his support, encouragement and patience, it was not possible for me to get stuck with the research. With his sharpness and attention to detail, Dr. Koen V. Hindriks provided useful feedback on how to present ideas and write papers. Both of them were very critical about my work, which made me feel uneasy sometimes, but eventually I know it was good for me, and now I am also very critical about my students' work.

I also want to thank Dr. Pascal Wiggers, who supervised me during the first two years of my stay at TU Delft. He proposed many interesting ideas and introduced me into the field of developmental robotics.

It has been a great experience to work with all my colleagues in the Interactive Intelligence Group. I enjoyed the lunch times, parties, colloquiums, days out, no matter it's a lovely sunny day or a terrible stormy day. I have opened my mind wide and got to know more of the world through you guys.

Thanks to all my Chinese friends in the Netherlands with whom I played table tennis, drank & brewed beer, and I traveled. Without you, it would not have been possible for me to keep healthy and strong during those years.

I also wish to thank my family for their support, especially my mother, who always misses me so much, but never says so.

Finally, I apologize that I couldn't mention all your names here. But I know that you know that I thank you all for your support during my days in Delft. Those are my most precious memories till now.

Chang Wang
Changsha, China

Contents

Chapter One

Introduction

This chapter introduces the research background, research questions and an outline of the dissertation.

1.1 Service Robots

In the near future, service robots are likely to share with humans household environments and assist all kinds of human activities such as preparing breakfast, cleaning the house, taking care of children, assisting elderly people, etc. Besides understanding what humans exactly want, robots should also be able to autonomously perform actions to achieve desired goals. Consider tasks like opening and closing refrigerator or oven doors in order to put in or take out food, pouring water into a cup or a bowl to make tea or cereal, and so on. Such tasks usually involve household objects which are composed of several parts and are designed for a specific use. Therefore, robots should be able to obtain knowledge about these objects and develop skills in order to handle them properly.

Manual programming of such knowledge and skills is only possible in carefully designed settings. For example, in the case of manufacturing environments, software developers program action commands for industrial robots to manipulate objects in a desired way. Robot arms can be preprogrammed to pick and place parts in assembly lines. In such environments, robots can repeatedly perform the actions without any changes. However, this would hardly work for service robots that are expected to solve a range of household tasks. First, household environments are usually unstructured, complex, dynamic and partially unknown. It is unrealistic for software developers to program perfect robot behaviors for all kinds of tasks at the design time. In addition, household robot users usually do not have the programming skills of robot developers. Therefore, a household robot, once deployed, should be able to adapt to a new environment through exploration as well as through natural interaction with the users. For these reasons, the learning capability is essential for service robots to develop new skills and knowledge.

1.2 State-of-the-art

Designing intelligent robots that can learn by themselves has taken inspiration from theories of neuroscience and psychology. The field of developmental robotics [1, 2, 3, 4] has followed this path and became a paradigm for developing cognitive robots that can acquire increasingly complex skills and competences. It is characterized by task-independent learning mechanisms as well as open-ended development during long-term embodied robot interactions with environments. Specifically, embodiment emphasizes the importance of a physical body which enables information structuring for developing cognitive functions [5]. These functions include the discovery of body dynamics such as hand-eye coordination, locomotion, and object manipulation. Currently, these are still open challenges in robotics.

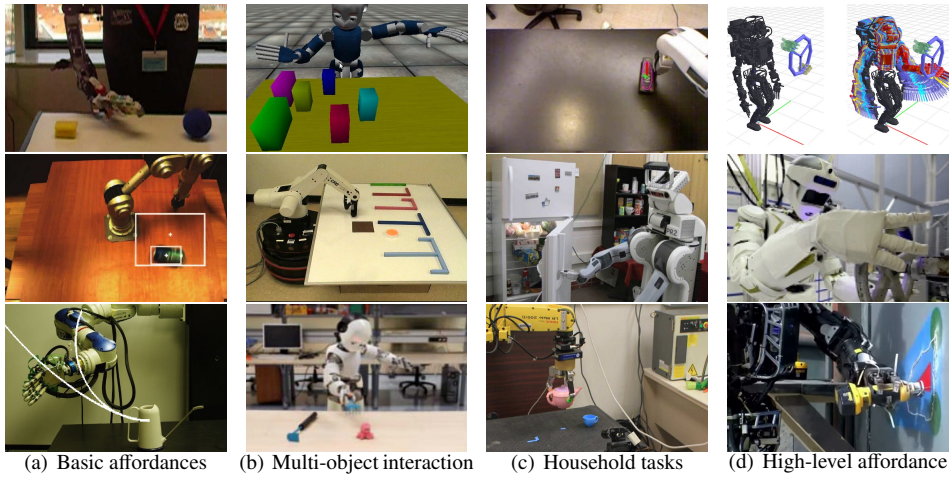
Affordance in Robotics

One important topic in developmental robotics is learning object affordances. The concept of affordance originates from the field of psychology [6]. It describes the relation between an organism and its environment that affords the opportunity for an organism to perform actions. For example, a door handle affords rotation by hand, a pedal affords pressing down by foot, and a stair riser affords climbing [7]. Neurophysiological evidence has proved that human perception of objects automatically suggests actions that can be applied on the objects [8]. This point of view has provided meaningful insights into the integration of perception and action for developing artificial cognition.

Such integration has been addressed by modeling affordances as the relations between objects, robot actions and the consequent effects [9, 10]. Affordances capture the distinctive features of objects in terms of what can be done with them. In other words, affordances provide information about potential action effects on objects, and this information can then be used to select actions to achieve task goals. In a kitchen scenario, objects with sharp edges are likely to be used for cutting food, objects with hollow parts are likely to be used as containers, and handle-like parts are likely to be grasped and pulled. Furthermore, affordances are task-independent, so that they can be reused across a range of tasks. For example, container-like objects can be used either for preparing cereal, or for preserving left-overs in refrigerators. Therefore, object affordances are useful knowledge for service robots to acquire.

In the literature, the use of affordances has been demonstrated beneficial compared with alternative methods that do not use affordances. For example, considering affordances can improve object recognition [11, 12] as well as human activities recognition [13, 14]. In the scenario of language learning, the affordance-based approach predicts adjectives with higher accuracy than the appearance-based approach [15]. In [16], the complexity of state space representation can be reduced by using affordances. In a real world environment, the navigation task can be learned faster for the affordance-based approach than the model-based approach [17].

In the sequel, a brief review is given to provide insights on the research trends on affordances for robotics. A more detailed literature survey will be given in Chapter 2.



(a) Basic affordances (b) Multi-object interaction (c) Household tasks (d) High-level affordance

Figure 1.1: Affordances in robotics. (a) Basic affordances are learned by a single pushing or grasping action [10, 20, 29]. (b) Affordance learning involves the interaction of multiple objects [35, 36, 37]. (c) In cluttered household environments, robots learn how to push and orient objects [38], open doors [13] and serve tea [14]. (d) High-level affordance knowledge is manually provided for object manipulation [39, 40, 41].

Basic Affordances

As the first step, single objects have been used to learn basic affordances (see Figure 1.1(a)). The objects are characterized by their shapes. The basic affordances include liftability [18], rollability [19, 20], pushability [21], traversability [22, 23, 17], and graspability [24, 25, 26, 27, 28, 29, 30]. In case of learning object movability, round objects have been found to move further than the cubic ones by applying the same pushing action on them. In case of learning grasp affordances, the possibility of successful grasps can be estimated at various locations on objects. The learned affordances have been used for action selection in an imitation game [31, 10], as well as for planning a sequence of actions to achieve given task goals [32, 33]. The learned affordances can also be used for language learning, e.g., understanding adjectives and nouns such as “tall”, “short”, “box” and “ball” [15] or sentences such as “tapped ball rolls” [34].

Multi-object Interaction

Recent research has taken into account affordances that involve multi-object interaction (see Figure 1.1(b)). The affordance model introduced in [10] is combined with statistical relational learning (SRL) to learn relational affordances [35]. It allows the reuse of single-object affordance for learning two-object interaction as well as the generalization of two-object interaction to more object cases. Also extending the previous model [33], single-object affordance is first learned and then reused as an input to improve the prediction accuracy of multi-object stackability [42]. In [36, 43, 37, 44], tool affordances have been learned where tool objects are grasped by the robot in order to interact with other objects.

For example, the iCub robot learns to retrieve an unreachable target object by choosing a rack to get the object closer for grasping [37].

Household Tasks

Another trend is to develop affordance-based robot behaviors in household tasks rather than in laboratory settings (see Figure 1.1(c)). In a cluttered environment [45], the robot recognizes the table surface and the target object on it, e.g., a television remote control or a dinner bowl. Then, different pushing strategies are systematically evaluated for positioning the target object [38]. In another work, human activities and object affordances are learned from RGB-D videos [13]. Reactive robot behaviors are developed to anticipate human activities based on spatio-temporal trajectories [46]. For example, the robot would open the refrigerator door if it observed a person holding an object and walking towards a fridge. In a multi-step tea service task, the robot learns to deal with a teabag, a kettle and a cup. The learning is based on human demonstration [47]. The robot learns dynamic motor primitives (DMP) [48] as well as object affordances in a unified skill learning and inference framework [49].

High-level Affordance Knowledge

By incorporating human guidance, robots can immediately acquire high-level affordance knowledge for complex task execution, e.g., turning wheels and drilling the wall (see Figure 1.1(d)). In the scenario of DARPA Robotics Challenge (DRC)[1], objects of interest are associated with task-specific goals [39]. In this way, the robot interacts with the environment at the level of affordance knowledge instead of traditional low-level teleoperation. Similarly, the concept of Affordance Template (AT) is proposed in [40, 41]. The AT framework follows a supervisory control paradigm of Plan, Teach, Monitor, Intervene, and Learn (PTMIL) [50], although the presented work does not support the Plan and Learn steps yet. In the scenario of Robocup@Home[2], functional affordances have been modeled in Description Logics (DL) for planning goal-directed tasks to assist human activities [51]. In case that the human requested for drinking tea, the robot would search for a teabag, a teacup and a kettle with hot water. If the teacup was not available, the robot would suggest substitute it with a mug which shared similar functional affordance with a teacup.

1.3 Challenges

Previous research has progressed from learning basic affordances to learning multi-object interaction, from laboratory settings to household environments, and from sensory-motor level affordance learning to high-level reasoning with affordance knowledge. However, several challenges still remain. In this dissertation, we address four challenges with regard to affordance learning and use.

[1] http://www.theroboticschallenge.org/
[2] http://www.robocupathome.org/

Goal-directed Affordance Learning and Use

Many approaches are based on a staged development framework [19, 32, 22, 33, 23, 31, 10, 15, 29, 35, 42, 36, 43, 37, 44]. A robot would first go through a goal-free "motor babbling" [19] stage, i.e., random action selection for a predefined number of trials. This is basically an exercise with the sole purpose of collecting training data. Thereafter, affordance models are learned by using the training data. In the second stage, the affordance models are used to select actions to achieve goals. However, the staged framework separates affordance learning and affordance use, so that there is no learning of affordances in the latter phase. As a result, the staged framework has difficulties in handling the situations that previously learned affordances do not hold any more when the affordances are used to select goal-directed actions. For example, when the robot has learned that a box is pushable during the training stage, it might end up repetitively trying to push the box even if the box is pushed against a wall. It is a challenge to design a mechanism that allows the robot to select reasonable goal-directed actions through affordance learning and use.

Exploration in Continuous Action Spaces

Discrete robot actions have been assumed effective for manipulating objects in well designed environments [19, 32, 52, 22, 33, 35, 18, 27, 20, 23, 44, 42, 53]. This assumption would hardly hold in general household settings where successful manipulation requires the corresponding actions to be defined in continuous spaces. For example, drawers, cabinets, ovens or refrigerators come with different designs. Predefined discrete actions can easily fail opening them in order to put in or take out other objects. Therefore, a robot should be able to learn effective manipulation skills through self-exploration in continuous action spaces. However, a continuous action space provides infinitely many action choices which makes random exploration time consuming. It is a challenge to find an efficient exploration strategy for object manipulation in continuous action spaces.

Transfer of Learned Affordances

Learning affordances from scratch is not efficient because it takes time for an embodied robot to perform exploratory actions on objects. Transferring learned affordances of known objects may speed up the learning of a new object. In the literature, knowledge transfer has been considered for effect prediction and action planning [33, 35, 54]. For instance, round objects are all likely to roll by a pushing action. However, it has been assumed that the source of knowledge transfer is known, and the previously effective actions are still effective for similar objects. In other words, no transfer failure is anticipated, and the robot would not recover from a failure and update its action selection strategy accordingly. Besides, it is even more difficult to handle household objects that usually consist of several parts than to handle simple toy objects. Therefore, it is a challenge for the robot to select by itself the relevant objects as the knowledge transfer source, evaluate by itself the actual outcomes of the transfer, and adjust by itself its action selection strategy if necessary. For example, assume that the robot has learned how to pull open a refrigerator and push open a door, it should be able to figure out how to open an oven by using its previous experience.

Integration of Symbolic Reasoning and Affordance Learning

Service robots are expected to solve a range of real world tasks through learning and using object affordances. Such a task can be characterized as achieving a goal through executing several subtasks and a sequence of actions. In the case that the robot is asked to get a bottle of beer from the refrigerator, it has to navigate to the refrigerator, open the refrigerator, grasp the beer bottle, take it out, and close the refrigerator. This kind of task domain knowledge is usually encoded in a symbolic form [39, 40, 41, 51]. This is because the symbolic representation is convenient for high-level reasoning and task planning. Meanwhile, affordances are learned from sensory-motor experience during embodied robot interaction with environments. It is a challenge to bridge the gap between high-level symbolic reasoning and sensory-motor level affordance learning. On one hand, learned affordances should be represented in a symbolic form and used for reasoning. On the other hand, the symbolic affordance knowledge must be verified by the robot through its own actions.

Each of these challenges will be addressed in detail in the dissertation.

1.4 Research Objectives

The overall aim of this dissertation is to improve the task performance of a robot through affordance learning and use. The main research question is:

"*How can object affordances be learned and used efficiently by an embodied robot in order to improve its performance for solving goal-directed tasks?*"

As a basic requirement, the robot must have a body that allows sensory-motor interaction with objects. The learning and the use of affordances are mentioned together because the learned affordances would be useless if they were not used for action selection to achieve goals in tasks. Specifically, the efficiency of affordance learning is important due to that the learning is usually time consuming. We investigate how to speed up the learning of affordances as well as how to use the learned affordances to improve task execution.

In the sequel, we propose four sub-questions of the main research question that are corresponding with the four challenges discussed in section 1.3.

First of all, the research is focused on designing an integration of affordance learning and affordance use rather than following the staged framework [19]. It results in the following sub-question:

(i) How can affordances be learned and used on-line for solving a goal-directed task?

Second, the research is focused on efficient affordance learning in continuous action spaces, i.e., to enable autonomous learning without manual coding of discrete actions as solutions, which results in the following sub-question:

(ii) How can a robot explore efficiently in continuous action spaces to learn affordances of a new object?

Third, assume that the robot is able to learn object affordances and use them to achieve

task goals. Nevertheless, learning every object from scratch would not be necessary if the robot had already obtained some knowledge about other objects. This takes into account long-term robot interaction with environments. In this way, the performance of solving the task at hand could be improved by reusing previously learned affordances of relevant objects. Therefore, the following sub-question is considered:

(iii) How can the learned affordances be transferred across objects to speed up the learning of a new object?

Finally, symbolic knowledge representation and reasoning play an important role of solving complex household tasks. Meanwhile, affordance learning is based on sensory-motor experience of the robot. The need of their integration results in the following sub-question:

(iv) How can affordance learning be integrated with symbolic reasoning for solving complex tasks?

1.5 Dissertation Outline

In accordance with the four sub-questions outlined above, the remainder of the dissertation presents the corresponding chapters to answer these questions. An overview of this dissertation is presented in Figure 1.2.

Chapter 2 first gives a brief introduction of machine learning techniques that are related to robot learning of affordances. Then, a literature survey shows how sensory and motor skills have been developed for robots along with how affordances have been learned by robots. The survey also provides insights on how affordances should be learned and used to improve robot performance for solving goal-directed tasks. Based on the insights, the following chapters will propose four learning architectures with increasing difficulties as the main contributions of this dissertation.

In Chapter 3, we propose the first architecture that deals with discrete object state and robot action spaces (based on publication [55]). The architecture integrates simultaneously on-line learning and the use of affordances in a reinforcement learning (RL) [56] framework. Affordances are stored as interpretable triples in a table that can be updated and reused across tasks. More specifically, affordances are acquired automatically during on-line task learning whenever an action is performed. But, while being learned, they are also used for action selection in solving the learning task at hand. In other words, we pay special attention to the on-line use of affordances as well as on-line learning of affordances. This distinguishes our approach from the approaches following the staged development framework [19, 32, 22, 33, 23, 31, 10, 15, 29, 35, 42, 36, 43, 37, 44].

Chapter 4 extends the discrete state and action spaces in Chapter 3 to continuous spaces. In other words, affordances are to be learned in continuous state and action spaces which contain infinitely many data points. This makes the training data collection and affordance learning much more difficult than in the discrete case. In purpose of collecting training data efficiently, an active affordance learning architecture is introduced (based on publi-

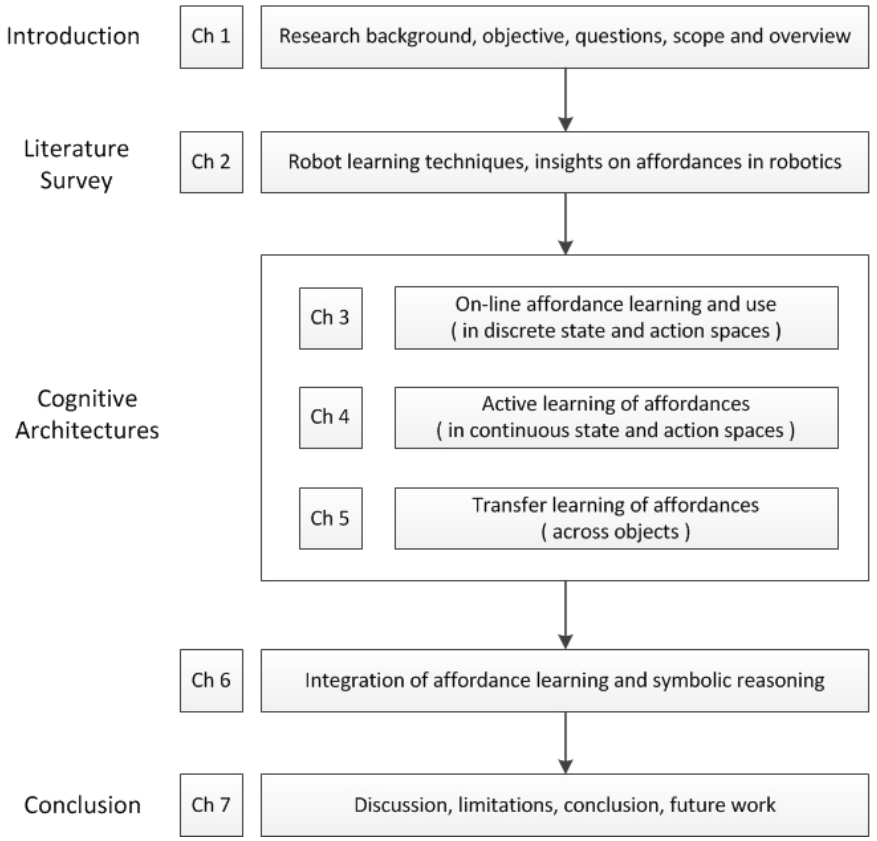


Figure 1.2: Dissertation overview

cation [57]). In this architecture, an action selection module actively decides what action is to be performed according to the observed object state. After an action is performed, the corresponding training data are collected to learn affordances, and function approximation is used to generalize over the continuous spaces. Simultaneously, the action selection module is updated. Compared with other approaches that have also addressed affordance learning in continuous action spaces, our active learning architecture improves action exploration efficiency over random motor babbling [43, 28, 37]. Besides, the architecture enables autonomous motor skill acquisition in a bottom-up manner in contrast to the traditional top-down manner, e.g., based on human demonstration [49, 14].

In Chapter 5, we further extend affordance learning of individual objects of Chapter 4 to a multi-object scenario. A transfer learning architecture is proposed to make full use of the robot's past sensory-motor experience to improve its long-term task performance (based on publication [58]). It aims at speeding up the learning of a new object through knowledge transfer from relevant known objects. Object relevance is measured by comparing object features such as shape and spatial relations between object parts. As a result, the robot is able to select by itself the source of knowledge transfer rather than decided by a human as it is done in the literature [33, 35, 54]. Then, the actual action effects are taken into account to verify the prediction of an anticipated knowledge transfer. This allows the robot to revise its action selection strategy if the transfer fails. In other words, it prevents the robot from being stuck with a wrong action decision, e.g., keep pushing a pull-door which looks like a push-door. This is an improvement over previous approaches which made predictions of the transfer results without any verification [18, 12].

Chapter 6 proposes a cognitive robot control architecture that subsumes the three architectures proposed in the previous chapters. The cognitive architecture integrates affordance learning with symbolic reasoning so that more complex tasks can be solved through affordance learning and use. Using the GOAL agent programming language[3], the cognitive architecture enables the robot to keep track of its goals, beliefs, and the affordance knowledge to structure its decision-making. First, the symbolic representation of affordance knowledge in GOAL facilitates affordance-aware action selection to solve goal-directed tasks. Second, the symbolic affordance knowledge can be verified by the robot through its own actions. Third, affordance learning can be initiated autonomously by the robot under certain conditions. These key features distinguish our cognitive robot control architecture from other approaches in the literature [39, 40, 41, 51, 59].

Finally, Chapter 7 answers the main research questions and concludes the dissertation by presenting the main contributions, limitations and suggestions for future work.

[3] http://mmi.tudelft.nl/trac/goal

Chapter Two
Robot Learning

This chapter gives an overview of robot learning approaches that are relevant to affordance learning. After a short introduction of machine learning techniques (section 2.1), the literature on affordance learning is reviewed (section 2.2). Then, further insights are given in section 2.3.

2.1 Machine Learning

Situated in a physical environment, an embodied robot interacts with the environment by processing sensory data and executing motor commands. For simple task settings, the motor commands can be preprogrammed by humans. However, this would hardly work in complex and dynamic environments where robot learning is necessary for developing adaptive robot behaviors. Typically, machine learning techniques play a key role in the generalization of sensory and motor experience, for accurate prediction and reliable decision making. This generalization ability enables a machine learner to handle new situations, e.g., learning to grasp a novel object or to navigate in a new environment. In the sequel, we introduce machine learning basics that are relevant to the robot learning tasks in this dissertation.

2.1.1 Supervised Learning

Supervised learning deals with the problem of learning a mapping from a set of *labeled* training data [60]. The training data are represented as paired input-output examples $\{(x_1, y_1), ..., (x_N, y_N)\}$, each input x_k corresponds to a desired output y_k $(k = 1, 2, ..., N)$. The input x_k is often represented as a feature vector in an input space X, and the output y_k is typically a known label or a measured value in an output space Y. Then, a supervised learning algorithm is used to produce a function $G : X \rightarrow Y$, where G is a mapping from the input space X to the output space Y. When a new input $x \in X$ is presented, the function G is expected to generate an accurate prediction $G(x) \in Y$. The prediction accuracy of G can be evaluated by a set of testing data. In addition, optimization can be carried out to find an optimal function G among a set of possible candidates. Refer to [61] for a detailed discussion about the representation, evaluation and optimization of learning algorithms.

The choice of a learning algorithm depends on the actual learning problem at hand. Popular supervised learning algorithms include Support Vector Machines (SVMs) [62], Artificial Neural Networks (ANNs) [63], Linear Regression (LR) [64], Gaussian Processes (GP) [65], Nearest Neighbor (NN) [66], Decision Trees (DTs) [67], etc. They have been widely employed in classification and regression tasks. In the sequel, we briefly introduce two examples of supervised learning tasks that are closely related to this dissertation.

Classification tasks involve the problem of assigning a discrete value, i.e., a class label, to an observation. For example, an SVM classifier can be constructed to classify images as door or non-door in a robot navigation task [68].

Regression tasks usually deal with function approximation problems in continuous spaces, which require no human labeling or discretization as in classification tasks. For example, ANNs can be used to approximate the relations between object states, robot actions and the consequent effects in continuous state and action spaces (see Chapter 4).

2.1.2 Unsupervised Learning

Unsupervised learning attempts to find hidden structures in *unlabeled* data. In analogy to the classification tasks mentioned above, clustering is an unsupervised learning task that groups unlabeled data samples. Based on a chosen criterion (e.g., distance measure in the data space), data samples in the same group (i.e. cluster) are expected to be more similar (or closer) to each other than to those in other groups. For example, k-means [69] is a popular clustering method that groups n observations into k clusters ($k \leq n$). Each cluster has a centroid so that each observation belongs to the cluster with the centroid closest to the observation.

2.1.3 Reinforcement Learning

Reinforcement Learning (RL) addresses task learning through direct interaction between an agent and its environment. The task is formulated in terms of states, actions and rewards. The RL agent learns to make optimal decisions, i.e., selects actions in given states to maximize its cumulative rewards received from the environment.

An RL agent in an embodied robot also makes action decisions (see Figure 2.1). The sensors and effectors of the robot are responsible for the interaction between its internal environment and its external environment. The internal environment is manually programmed. It not only processes the sensations from the external environment (real world) into a representation of states, but also sends action commands to robot effectors for execution[1]. Most importantly, the internal environment contains the reward function that provides rewards to the RL agent. In a goal-directed RL task, the reward function is always related to the given goal, e.g., a destination in a navigation task (see chapter 3). It can also be related to intrinsic motivation [70], e.g., a prediction error (see chapter 4).

RL algorithms are used to find solutions for RL agents in Markov Decision Processes (MDPs). An MDP is formulated as a four-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$. We now discuss in more detail each of the four components of an MDP. Refer to [56] for a detailed overview of reinforcement learning algorithms.

[1] In this dissertation, it is assumed that the robot can always execute the actions decided by the RL agent.

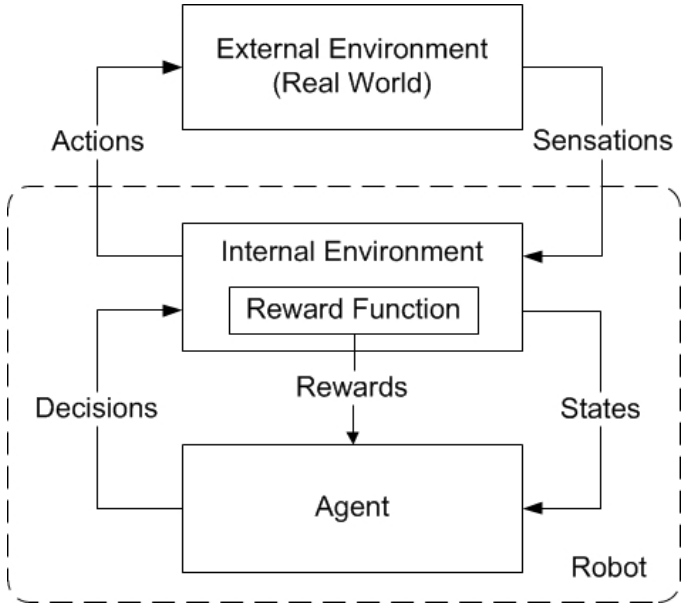


Figure 2.1: Reinforcement learning for an embodied robot (based on [71]).

State describes the environment features that are relevant for a learning task. Formulating an appropriate state representation is essential for the performance of the RL agent. In other words, the RL agent cannot make good decisions without sufficient information about the environment. With too many irrelevant details about the environment, the problem complexity would increase . The feature selection for state representation is usually done by a human expert before the RL agent starts learning. In RL systems, discrete time steps are used. At the time step t, the state is denoted by $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of all possible states. For example, s_t can include low-level features such as raw sensor readings, or high-level features such as interpreted readings, as well as other relevant information about the task at hand.

Action is the way for the agent to influence its environment. The output of the RL agent is an action decision to be executed in a given state. It can be a discrete value from a predefined set, or a continuous value within a predefined range. For example, a discrete action can be "walk 5 cm forward". In the case of continuous actions, the RL agent has to learn a continuous mapping from states to actions, rather than choose an action from a discrete set. At time step t, the agent uses a policy π to calculate action $a_t \in \mathcal{A}$ in state s_t, where $\mathcal{A}$ is the set of all possible actions. The action a_t leads to a state transition in the environment: $s_{t+1} = \mathcal{P}(s_t, a_t)$, where $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the state transition function.

Reward shapes the behavior of an RL agent. In a goal-directed task, a positive reward can be given to encourage state transitions toward the goal state, e.g., reaching the destination in a navigation task. In addition, negative rewards can be given to avoid unwanted states, e.g., when colliding with obstacles during navigation. The rewards come from a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ that maps a state s_t, an action a_t and the consequent

state s_{t+1} into a reward $r_{t+1} = \mathcal{R}(s_t, a_t, s_{t+1})$. The agent aims to maximize the discounted sum of rewards, which is also known as the return. The discounted return of a policy π is given by the expectation $\mathrm{E}\Big\{ \sum_{i=0}^{\infty} \gamma^i r_{i+1} \Big| d_0, \pi \Big\}$, where $0 \leq \gamma < 1$ is the discount factor and d_0 is the distribution of the initial state s_0. Values of a state s or a state-action pair (s, a) are stored to estimate the return. The value function $V^\pi : \mathcal{S} \rightarrow \mathbb{R}$ captures the expected return when starting in a state s under policy π:

$$V^\pi(s) = \mathrm{E}\Big\{ \sum_{i=0}^{\infty} \gamma^i r_{i+1} \Big| s_0 = s, \pi \Big\}, \tag{2.1}$$

and the Q-function $Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ captures the expected return received after taking action a in state s and following the policy π afterwards:

$$Q^\pi(s, a) = \mathrm{E}\Big\{ \sum_{i=0}^{\infty} \gamma^i r_{i+1} \Big| s_0 = s, a_0 = a, \pi \Big\}. \tag{2.2}$$

Both functions satisfy the Bellman equation [56] and they have the following relation:

$$Q^\pi(s, a) = \mathcal{R}(s, a, s') + \gamma V^\pi(s') \tag{2.3}$$

where $s' = \mathcal{P}(s, a)$ is the next state of s. The optimal policy π^* satisfies

$$\pi^* = \arg\max_\pi V^\pi(s), \forall s \in \mathcal{S} \tag{2.4}$$

and corresponds to the highest possible return

$$V^*(s) = \max_\pi V^\pi(s), \forall s \in \mathcal{S} \tag{2.5}$$

and

$$Q^*(s, a) = \max_\pi Q^\pi(s, a), \forall (s, a) \in \mathcal{S} \times \mathcal{A}. \tag{2.6}$$

Learning these functions is at the core of reinforcement learning. Details of specific RL algorithms will be given in the corresponding chapters. For the ease of notation, we drop the superscript π in the rest of this dissertation.

2.2 Affordance Learning in Robotics

2.2.1 Background

The concept of affordance was originally proposed by Gibson in the field of ecological psychology [6]. Based on the studies of visual perception, Gibson defined affordance as action possibilities offered to an organism by an environment. For example, if a stair riser is less than a certain percentage of a person's leg length, it means that the person can climb that stair [7]. Such a relation has been believed to be *directly perceived* by the organism in the environment . Following Gibson's work, there have been a number of studies which offer refinements of the affordance definition, such as [72, 73, 74, 75].

However, it may require some flexibility on the definition of affordance to benefit the field of robotics, as discussed in the survey [76]. Refer to [6, 77] for further reading about affordance in ecological psychology, and refer to [78] for computational models related to affordance.

2.2.2 Affordance Definitions in Robotics

There are three different perspectives to view and define affordances [9], i.e., human observer perspective, robot perspective, and environmental perspective (see Figure 2.2).

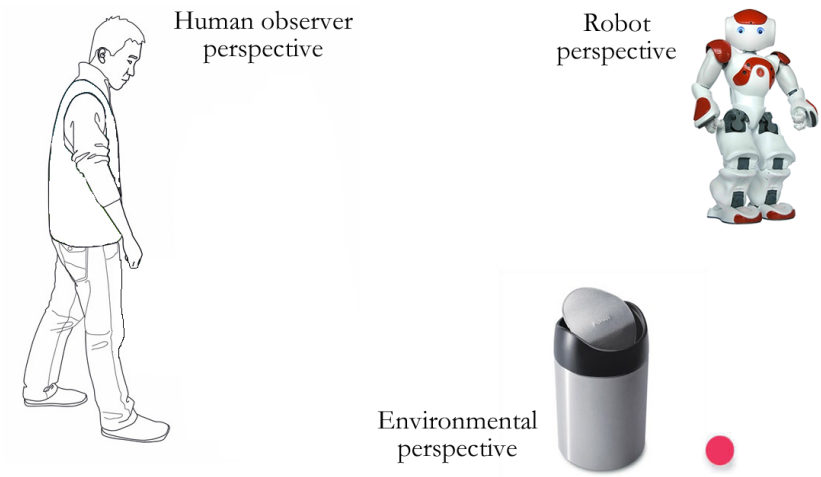


Figure 2.2: Three perspectives to view affordances in the human-robot-environment system. In this scene, a robot is expected to pick up the red ball and drop it into the garbage can. This interaction is being observed by a human.

In the human observer perspective, affordances in the robot-environment system have been believed to be perceivable by the human. In the case of Figure 2.2, the human would say:"There is graspability affordance" in the robot-ball system. We note that the human observer is invisible to the robot, and human-robot interaction is typically not considered.

In the robot perspective, the robot interacts with the environment and discovers the affordances through its own actions. This view is the focus for robot learning of affordances in this dissertation. The robot actions are usually programmed and tasks are defined to establish whether the robot actions are suitable for affordance-based perception [79]. In Figure 2.2, the robot would learn by itself whether there exists graspability after trying to grasp the red ball.

In the environmental perspective, affordances are simply regarded as extended properties of the environment. For example, the garbage can offers liftability (to the human) and pushability (to both the human and the robot).

In some cases, an affordance is defined as a description of an *effect* (e.g., traversed,

rolling, grasped, lifted and stacked) which is both observable to the human observer and the robot. The effect typically takes a binary value which is true or not true. The corresponding affordances are called traversability, rollability, graspability, liftability and stackability (see Table 2.1). This way of defining affordances takes into account only a single robot action and its effect at a time. The focus is on the prediction of the effect. However, the robot action is not included in the affordance definition.

Table 2.1: A summary of affordance definitions.

Affordance Definitions	Related Papers	Explanation
Traversability	[22, 23, 80, 81]	The robot predicts whether it is able to traverse or not the given environment.
Rollability	[19, 20]	Round objects would roll and cubic ones would not after being pushed or poked.
Graspability	[27, 29]	The robot predicts whether a given object is graspable or not.
Liftability	[18]	Objects are classified as liftable or not liftable based on their shapes and colors.
Stackability	[42, 53]	The rollability of single objects is used to predict the stackability of paired objects.
Goal-based	[36, 38, 39, 41]	Emphasizes the importance of both objects and actions to achieve goals.
One-directional model	[9, 33, 37, 49, 59]	It maps from a pair of object and action to an effect. It supports both effect prediction and action selection.
Bi-directional model	[10, 35, 44]	It is based on a network structure with connected nodes. It supports both effect prediction and action selection.

In contrast, the goal-based definition of affordance emphasizes the importance of not only object properties, but also robot actions to achieve goals. In the scenario of tool use, the tool's affordances are defined in terms of robot actions and the statistics of goal achievement, which are saved in an *affordance table* [36]. In [38], the task goal is to push and orient any given object. An affordance is defined to exist between a robot and an object if the robot can successfully perform the desired action to achieve the goal. In [39, 41], affordance templates (AT) are defined for goal-directed object manipulation tasks. For example, a wheel-turning template is defined for turning valves in the scenario of DARPA Robotics Challenge. Different from the affordance definitions in the robot perspective, the AT approach takes the environmental perspective and involves a human operator that tele-operates the robot.

Considering both the effects and actions, two kinds of affordance models have been defined (see Figure 2.3). One of them models an affordance as the one-directional relation

between an (*entity*, *behavior*)[2] pair and an *effect* [9]. The other one models affordances as the bi-directional relations between *objects*, *actions* and *effects* [10].

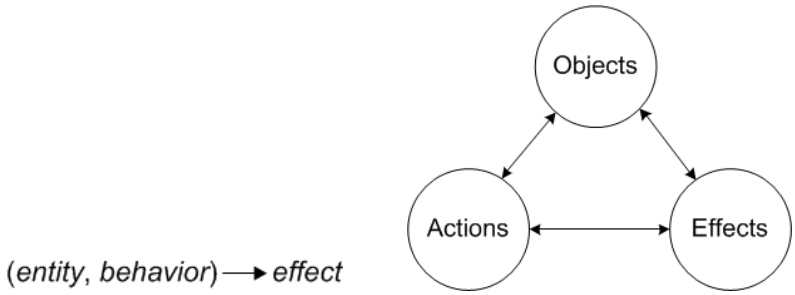


Figure 2.3: One-directional and bi-directional affordance models.

In general, the one-directional model maps from a pair of object and action to an effect. However, the formalisms of one-directional affordance models vary in the literature. In [33], the robot arm interacts with several objects by using three push actions and one lift action. Each action is bound with a model that predicts the effect of object movement. Similarly, the movement of a toy car is predicted after being pushed in various directions [37]. In [49], an affordance is represented as a tuple of a precondition (object), an action and a postcondition (effect) to configure task-relevant objects in a multi-step tea service task. In [59], an Object-Action-Complex (OAC) captures the interaction between an object and the robot using a prediction function about state changes (effect) caused by the execution of a robot action.

In contrast, all the bi-directional models are based on a network representation which has a structure of connected nodes [10]. Typically, the nodes take discrete values. For example, the object nodes can represent objects by their colors (green, yellow), shapes (ball, box) and sizes (small, medium, big); an action node takes values of grasp, tap and touch; and an effect node describes object displacement (small, medium, big). Given the values of object and action nodes, the model can infer the most likely values of the effect nodes. Similarly, given the values of object and effect nodes, the model can infer the most likely values of the action nodes. In [35], the same representation is used and combined with statistical relational learning in a multi-object scenario. In [44], the network is extended with nodes of "tool objects" in addition to the nodes of objects, actions and effects. Then, intermediate object affordances are modeled towards the development of a tool concept.

2.2.3 Sensory Functions and Motor Skills

As a prerequisite for affordance learning, sensory functions and motor skills have been assumed available for a robot. In other words, the robot can recognize objects, perform

[2]The term *entity* refers to the perceptual representation of an object. It has been assumed that an object can be segmented from the environment by computer vision algorithms. The term *behavior* denotes the executed perception-action routine that enables embodied robot interaction with the environment [9]. In this dissertation, we use *object* instead of *entity*, and *action* instead of *behavior*.

elementary motor tasks, and obtains the consequent effects. In this section, we review how these sensory functions and motor skills have been prepared for robots before introducing affordance learning in the next section.

Object recognition

Object recognition is carried out through the extraction of object features. For example, color and shape features can be extracted from camera images. The tracking of the features enables the tracking of objects. This provides relevant information about object states, e.g., object position, orientation and speed in the world space. The representation of objects can be classified into three types: fixed, discrete or continuous (see Table 2.2).

Table 2.2: Types of object representations for affordance learning.

Object representation	Related papers
Fixed	[19, 36, 37]
Discrete	[10, 18, 27, 35, 38, 44]
Continuous	[20, 21, 22, 23, 33, 43]

The fixed group only considers a fixed set of objects. In [19], four toy objects are recognized and tracked by color features. In [36], five tool objects are labeled and the affordances are saved in a look-up table. In [37], each object is segmented in 2D images and tracked using a bounding box. The states of the object can be updated. However, this representation considers only a specific object at a time. As a result, it has limitation for reusing past experience to handle new objects.

In order to classify given objects, the discrete group describes objects with a set of predefined labels. In [27], objects are described by shapes (cylinder, rectangle) and sizes (small, medium, large). In [18], object parts are segmented by color recognition and configured by top/bottom relations. In [10, 35, 44], the X-means [82] algorithm classifies colors based on histograms, while shapes are classified by local shape descriptors such as convexity, circularness and squareness.

Without prior discretization, the continuous group directly processes raw sensory data. In [22, 33], distance and local shape features are extracted from depth sensors. In [23], an object is described by distance features extracted from ultrasonic point clouds. In [21], object contours and shape gradient features are extracted. In [20], continuous sensory data is handled by a codebook vector layer with Euclidean distance metric. In [43], novel tools are compared with a fixed set of tools based on local shape features.

In summary, the complexity of object representation increases from the fixed group to the discrete and continuous groups. In this dissertation, we choose the object representations according to the research objectives in the corresponding chapters. Typically, objects are segmented and tracked in 2D images, and object states are represented in discrete or continuous spaces with consideration of new object states.

Motor skills

The motor skills for learning affordances are usually programed as a set of discrete actions or defined in continuous action spaces (see Table 2.3).

Table 2.3: Types of robot actions for affordance learning.

Action Type	Related papers
Discrete	[18, 19, 20, 22, 33, 35, 27, 23, 44, 42, 52, 53, 10, 28]
Continuous	[21, 37, 38, 43]

Discrete actions have been assumed effective for affordance learning in well designed environments. In [18], a lifting action is used for a magnetizable effector. In [22, 23], mobile robots traverse in predefined directions with fixed distances. In [19, 52, 20, 33, 35], robot arms push in given trajectories. In [27], the robot hand grasps at a certain height. In [10, 28], grasping, tapping and touching actions are parameterized, but only discrete actions are used in the experiments.

However, the discrete actions have limitations in handling novel objects that come with unknown shapes and sizes. For example, pushing an object in a predefined direction would fail if the object spin, e,.g., a round bowl. In order to solve this problem, controllers are defined to continuously interpolate pushing direction to position and orient objects [21, 38]. In [43, 37], pushing parameters are chosen randomly from a uniform distribution.

In Table 2.3 that most of the approaches in the literature only consider discrete actions. One main reason is that continuous actions increase the difficulty for affordance learning. In other words, it is more challenging to choose an effective action parameter from a continuous space than to define a discrete action in a well designed environment. In this dissertation, we will address the challenge of using continuous actions for affordance learning in chapter 4-6.

Perception of Effects

The perception of effects is based on the representation of objects. Usually, effects have been assumed to be the results of a robot' own actions. An effect is obtained by tracking an object and calculating the state changes of the object before and after an action. Effects can be either predefined, or self-discovered by the robot (see Table 2.4).

Table 2.4: Types of effect representation for affordance learning.

Effect representation	Related papers
Predefined	[18, 19, 36, 10, 27, 81, 22, 35, 38, 37, 42]
Self-discovered	[20, 33, 43, 52, 83, 84]

A popular way to define effects is to use labels, e.g., lifted or not, traversed or not, grasped or not, stacked or not, etc. More detailed descriptions consider object displacement, orientation change or object motion. In contrast, self-discovered effects are obtained

as unlabeled clusters in continuous spaces via unsupervised learning. Specifically, the X-means algorithm [82] has been widely used in the literature. In this dissertation, we define effects as labels or values in continuous spaces according to the object representation in the corresponding chapters.

2.2.4 Affordance Learning

Based on the prepared sensory functions and motor skills, affordances can be learned through embodied robot interaction with objects. This has followed the principles in developmental robotics [1, 2, 3, 4]. The learning approaches depend on how affordances are defined. Various machine learning algorithms have been employed to learn affordances (see Table 2.5).

Table 2.5: A summary of affordance learning approaches.

Learning Types	Related papers	Learning methods
Table-based	[36, 38]	Table-lookup
	[18]	Q-Learning
Supervised	[22, 23]	Support Vector Machines
	[10, 81, 35, 49, 44]	Bayesian Networks
Unsupervised	[20]	Neural Networks
	[33, 42, 37, 83]	Support Vector Machines
	[43, 52]	Nearest Neighbors
	[84]	Decision Tree

In our classification, learning types are based on not only how affordances are defined, but also how the learning is influenced by human. If data is stored in a table, it is classified into the "Table-based" type. If the effects or affordances are manually labeled, the approaches are classified into the "supervised" type. If the effects are self-discovered by the robot, the approaches are classified into the "unsupervised" group, though the machine learning algorithms might be "supervised" ones.

Table-based affordance learning

In an early study [36], a fixed set of tool objects are assigned unique labels. Each object label is associated with a set of actions and the statistics of goal achievement. All the data is saved in an *affordance table* for future query. This approach is simple and straight forward, but it can not generalize the learned affordances to handle novel objects. Therefore, it has a limited learning ability.

In [38], the robot learns to achieve a goal by selecting a perceptual proxy, a behavior primitive and a controller. Every combination of the three components and the consequent result are saved in a table to evaluate the learning strategy. For example, a centroid/ellipsoidal perceptual proxy, an overhead/sweep push behavior, and the centroid alignment/spin-

correction controller are combined to learn how to push and position a television remote and a dinner bowl.

In [18], reinforcement learning is used to learn object liftability. Rewards are assigned to the visual features of objects according to the results of lifting them. Positive rewards are given to encourage successful lifting; otherwise, negative values are given. These values are saved in a table. In this way, the robot learns visual cues in a purposive manner that the cue like states with high values are considered as good opportunities for future interactions.

Supervised learning of affordances

In [22, 23], the robot performs a set of discrete actions and a set of training data is collected. The objects are represented by low-level shape features, and the effects are labeled as movable or not. Each action is bound with a Support Vector Machine (SVM) that predicts the effect labels based on the observed shape features. This approach is able to predict the movabilty of novel objects by using the learned SVM.

In [10], affordance models are represented as Bayesian Networks (BNs) that encode the bi-directional dependencies between objects, actions and effects. Since the learning is based on a probabilistic BN model, the approach is able to handle uncertainty, redundancy, and irrelevant information. In the training stage, the robot also observes the effects of its own actions on objects. The structure of a BN is estimated by the Markov Chain Monte Carlo (MCMC) method. In [35], the BN model is extended to handle multiple objects by integrating Statistical Relational Learning (SRL). One model is learned for inference of any number of objects, without suffering from the structure learning of several BNs and switching between BNs for inferences. In [44], the BN is extended by including "tool objects" in addition to non-tool objects. In [49], the BNs encode the relationship between precondition, motion primitive, and postcondition in order to configure task-relevant objects. Also using BNs, a Category-Affordance model has an intermediate layer of object categories to associate object features and predefined affordance labels [81].

Unsupervised learning of affordances

Unlike supervised learning of affordances, unsupervised approaches require no predefined labels for training affordance models. Typically, these models are one-directional.

In [20], a cross-modal neural network is constructed by two sensory modalities. The input modality encodes object shape features and the output modality encodes object motions caused by a pushing action. Both layers are fully connected to each other. The collected data is in the form of input-out pairs that are based on the co-occurrence of the two modalities. No manual labeling of data is needed. The learned neural network can predict the motion of an object when given its shape features.

In [33, 83], effect clusters are self-discovered by a robot using the X-means algorithm. Similar with [22], SVMs are trained to associate object features with the effect clusters. In [37], SVM regression is used to approximate the mapping from the continuous action space (arm pushing direction) to the continuous effect space (object displacement).

In [52], objects are categorized as container or non-container according to the object motion after pushing. First, the X-means algorithm is used to discover object similarities by clustering the shape feature vectors. Then, a Nearest Neighbor classifier is trained with

automatically generated labels. The classifier can classify novel objects as containers or non-containers when given their shape features. Similarly, a Nearest Neighbor classifier is trained to learn tool affordances [43].

In [84], the X-means algorithm is also used to obtain effect clusters, and a decision tree classifier is trained to classify tools. The classifier is used to discover similar tools that result in similar effects. The decision tree classifier has been found to generalize better than the Nearest Neighbor classifier.

2.3 Discussion

In the previous section, we have reviewed the affordance definitions, along with how the affordances can be learned. This section provides further insights of the literature review.

In the literature, affordance learning has been focused on associating object features with the effects of actions. The main purpose is to discover the distinctive features that are likely to result in the effects. In addition, learned affordances can also be used to select actions to achieve goals. A typical example is that round objects are likely to roll while cubic ones are not. If a goal is given to stack objects, then the cubic ones are likely to be stacked [42, 53]. Novel objects can be classified not simply by their features, but by the potential effects that can be achieved.

However, the efficiency of affordance learning has been neglected in the literature. Typically, random actions are performed to collect data before machine learning algorithms are employed to learn affordances. This is not efficient because the learning is not even started until sufficient data is available. Moreover, it is the human operator who decides how much data is needed. This makes the learning even more inefficient, because the human can hardly interpret the sensory-motor data of the robot to decide whether the learning should be terminated or not. It is a challenge to improve the efficiency of data collection as well as to improve the efficiency of affordance learning based on the available data.

Besides, the need for open-ended affordance learning has been neglected. Once the learning stage is finished, the learned affordances are not updated anymore. It has been assumed that the environment is static and the affordances do not change between the environment and the robot (see Figure 2.2). Generally, this is not true in dynamic environments that the robot has to handle a variety of novel objects and situations. The learning should be enabled whenever necessary. It is a challenge for the robot to decide by itself when the learning is necessary.

In this dissertation, our focus is on efficient and open-ended affordance learning. Specifically, we aim to develop a novel framework that supports the following three learning mechanisms:

- *On-line learning*. This mechanism enables on-line data collection whenever an effective interaction happens between the robot and the objects. Also, the affordances are learned on-line with all the available data. As a result, the robot can keep on learning in dynamic environments in which the affordances may change. Chapter 3 proposes a table-based reinforcement learning framework for discrete state and action spaces. Chapter 4 and Chapter 5 discuss on-line affordance learning in continuous state and action spaces, which is more challenging than the discrete case. Chapter 6 not only

supports on-line affordance learning, but also the verification of learned affordances. Based on the verification result, affordance learning can be initiated and terminated by the robot itself.

- *Active learning*. This mechanism enables active selection of actions rather than random action selection. As a result, the data for affordance learning is collected actively by the robot itself. The learning is carried out in an unsupervised manner. While the affordances are learned, they are simultaneously used to adjust the action selection strategy. In this way, the data collection becomes efficient by incorporating useful information of the actual learning results. Chapter 4 will discuss active affordance learning in detail.

- *Transfer learning*. This mechanism transfers the learned affordances to speed up the learning of a novel task. A robot may encounter a variety of objects in dynamic environments. Learning each object from scratch is an option, but it would take a long time for the robot to try out every action on every object. It is more efficient to directly reuse the learned affordances to select actions on the novel object. Chapter 5 will discuss transfer learning of affordances in detail.

Finally, we note that the learning of affordances are closely related to the use of the learned affordances. The learned affordances not only can be used to select goal-directed actions, but also can be used to speed up the learning, as mentioned above in the cases of active learning and transfer learning. In the rest of the dissertation, we will discuss in detail the close relation between the learning of affordances and the use of affordances.

CHAPTER THREE

On-line Affordance Learning and Use in Goal-directed Tasks

A problem with regard to affordance learning and use in the literature is that the use of affordances is addressed separately from affordance learning. As a result, there is no learning of affordances while using them. On the other hand, goal-directed tasks can be formulated in the reinforcement learning (RL) framework, in which learned affordances are useful for action selection. This chapter proposes a cognitive robot learning architecture that supports simultaneous learning and the use of affordances in effective solving of RL tasks. We demonstrate the effectiveness of this approach by integrating affordances into an Extended Classifier System (XCS) for learning general rules in a RL framework. The experimental results show significant speedups in learning how a robot solves a given task.

3.1 Introduction

Many real world robotic tasks, like navigation or object manipulation, are dynamic and require on-line learning capability. 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, a *goal-directed task* specifies a goal state that has to be achieved. The robot interacts with objects in the environment, 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* [6] provides robots with information whether an object affords an action or not [9]. 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 chapter, the main goal and contribution is to investigate and propose a cognitive architecture that combines on-line learning of affordances and the use of affordances at the same time to improve the robot's learning performance in goal-directed tasks.

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 robot actions [79]. In the literature, some approaches considered a fixed set of robot actions for affordance learning under specific task settings (see section 2.2.3). For example, object movability is learned by the same pushing action [20]. Affordances are learned as mappings from the perception space of objects to the perception space of effects. The focus is on discovering relevant object features in the perception space, e.g., shape features, for making predictions of predefined effects. However, actions are not explicitly included in the representation of affordances. Therefore, these affordances cannot be used by robots for selecting or performing actions to achieve task goals.

Taking into account robot actions in addition to objects and effects, two major definitions have been proposed and applied in robotics (see Figure 2.3). Both approaches have followed the staged developmental framework of affordance learning and use. That is, a robot first collects training data by embodied interaction with objects and self-observation of the action effects. Then, the collected data is used to train affordance models by machine learning algorithms (see section 2.2.4). Afterwards, when a task goal is given, the robot makes use of the learned affordance models for action selection. For example, [32, 22, 33, 15, 42] use a one-directional affordance model. Each robot action is associated with a Support Vector Machine (SVM) that maps from the perception space of objects to the perception space of effects. When a set of objects and a goal effect are provided, the robot predicts what effects could be generated by each action on the given objects, and then plans a sequence of actions to generate the goal effect. In [31, 10, 35, 44], Bayesian Networks (BNs) are trained to capture the probabilistic dependencies between objects, actions and effects. When a goal effect is demonstrated in an imitation game, the robot is able to select an object and an action which are most likely to result in the goal effect [10, 31]. Also, the robot is able to select a tool object to interact with a target object in a goal-directed way [44].

In these approaches, however, affordance learning is carried out in an off-line learning manner after the collection of all training data. The learned affordances are believed to still hold in the stage of affordance use. This is not always true because affordances may change due to environmental changes. In order to address this problem, affordances have to be learned on-line when new data is available for the learning. This means that an on-line learning algorithm has to be employed for affordance learning. In addition, affordance learning should also be allowed when the robot makes use of learned affordances to plan and perform an action. Because after the action is performed, the robot observes the effect and collects new training data for affordance learning.

The main contribution of this chapter is the proposal of an architecture that integrates simultaneous on-line learning and 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. 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 [85], within a reinforcement learning [56] framework. In addition, we pay special attention to the on-line use of affordances. In contrast to previous approaches, we integrate affordance learning and the use in a unified perception and action loop. Our architecture

allows affordance learning while the robot performs a task, even in a dynamic environment.

The remainder of this chapter is organized as follows. Section 3.2 defines the affordance model for the rest of this dissertation. Section 3.3 proposes the cognitive architecture that allows the interaction between affordances and task learning. Then, sections 3.4 and 3.5 describe the robot platform and the task environments that we use in our experiments, respectively. Finally, Section 3.6 concludes the chapter.

Parts of this chapter have been published in [55].

3.2 Affordance Model

Similarly to [9, 10], we formalize affordance as the relation between an embodied robot and its environment.

Definition 3.1 An affordance is defined as the triple:

$$(Object, Action, Effect) \tag{3.1}$$

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 information is independent of the task at hand.

We now discuss in more detail each of the three components of an affordance.

3.2.1 Perception of Objects

A robot perceives its environment and extracts a set of features $\{f_i\}_{i=1}^{N}$ from its raw sensory input. Currently, many robots are equipped with RGB cameras or RGB-D cameras as the main source of visual perception. The extracted visual features can be at any level, e.g., from low-level KLT points [86], mid-level SURF [87] or SIFT [88] points to high-level category labels [89].

An object is denoted by $o \in \mathcal{O}$, where $\mathcal{O}$ denotes an *attribute space* which is defined as a collection of properties with assigned values [59]. For example, these properties can include object color, shape or size, and the associated values can be red/green/blue, roundness, or big/small. We note that the attributes can take continuous, discrete, or Boolean values for different problem spaces. We assume these values are *invariant* for an individual object. As a result, the size of the attribute space $\mathcal{O}$ decides the maximal number of different objects that can be recognized and represented by the robot. We also assume that the sensors can be used to extract the environmental state of an object which might be *changing* with time, denoted as $s_o \in \mathcal{S}_{obj}$, where $\mathcal{S}_{obj}$ is a state space. For example, s_o can be the current location of the object in the world space.

3.2.2 Robot Actions

Denote by $a \in \mathcal{A}$ a robot action and $\mathcal{A}$ is the action space. Robot actions can be defined in continuous spaces, e.g., in constrained joint space or in the Cartesian space with the as-

sociated inverse kinematics. They can also be programmed as a discrete set of behaviors. Generally, the continuous cases are more difficult than the discrete ones for affordance learning. This is because a continuous action space has infinitely many possible actions while a discrete set has a finite number of actions. Consequently, the relations between objects and actions are more complex for the continuous action case. The choice of an action representation depends on the robot platform and on the task at hand. In this chapter, we focus on discrete actions. In the next chapters, we will address the challenges of continuous actions.

First of all, we need to make some assumptions about robot actions. We assume that preconditions of actions are provided as domain knowledge. For example, an end-effector should reach the object before grasping it. In addition, we assume that the action commands defined in an action space can always be executed successfully by the robot. For example, executing the "open hand" command will not result in a "close hand" action. Otherwise, action failures would result in noisy training data which would hinder affordance learning. Although our learning approaches have taken into account noise in data, handling the noise is not the main focus of our research.

3.2.3 Perception of Effects

After performing an action a on the object o, the robot perceives a new state of the object $s'_o \in \mathcal{S}_{obj}$, where $\mathcal{S}_{obj}$ is the attribute space of object states. By comparing s'_o and s_o, the consequent action effect e is calculated and it takes a value in an effect space $\mathcal{E}$, which is also an attribute space. The effect e is obtained as follows:

$$e = m(s'_o, s_o) \tag{3.2}$$

where m is a suitable measure. For example, a box is pushed and its location change is measured in a real-valued continuous space. If a robot tries to open a door, the effect can also be a binary state description that the door is opened or not.

3.3 Cognitive Architecture

The affordance model (AM) defined in (3.1) shares a same component of *Action* with the formalism of an MDP (see section 2.1.3). The attribute space of *Object* also has a similar representation to the state space of an MDP. The differences between an AM and an MDP arise from how action effects are defined and associated with objects, and how actions are selected for robot learning. Effects in an MDP are described by a state transition function and correspond to a reward function. The reward function provides actual rewards and shapes the robot behaviors (a perception-action loop) by maximizing the expected accumulation of future rewards. This decision making manner considers a long-term consequence for action selection. In contrast, AM is usually learned from a set of training data which is collected via "motor babbling" (i.e., random action exploration). Then, AM is used to make an one-step prediction whether an object affords an action or not.

Based on the above comparison between an AM and an MDP, we propose that they can be coupled together to benefit each other. On the one hand, training data can be collected for affordance learning when an action is selected by the MDP and performed by the robot.

On the other hand, learned affordances can be used to aid action selection in the MDP. In this way, the learning and the use of affordances take place in the same loop while solving a goal-directed learning task that is modeled as an MDP.

3.3.1 Architecture Overview

The overall approach 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. Figure 3.1 illustrates the architecture we propose.

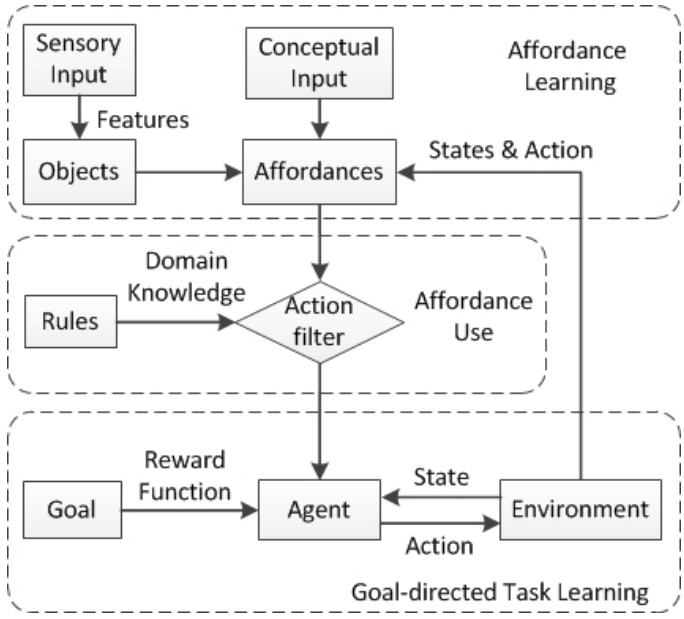


Figure 3.1: An architecture for affordance learning and use during task learning.

In the affordance learning component, the robot perceives the environment via its sensors. It extracts features and forms a representation of the environment (see section 3.2.1). The robot is provided with a set of actions to learn affordances and perform tasks. Additionally, the conceptual input provides the robot with the knowledge on which of the actions are related to affordance learning and how the effects of these actions are defined. For example, when learning object movability, a push action towards the object is relevant as well as a notion of distance and its change. Then, the robot can learn the movability of objects by observing their displacement after a pushing action. In other words, 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. The affordances are updated on-line during the task learning, by collecting data of the encountered object, the action performed and the resulting effect. 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 Figure 3.1, the learned affordances are used to filter out those actions which are likely to cause undesired effects. For this purpose, rules are provided as task domain knowledge to construct the action filter. These rules can be related to the current task goal, and common sense knowledge can also be included. As the general problem of relating the undesired effects with goals is beyond the scope of this chapter, we use a simple approach where we only use the common sense rules. We assume that if 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 Figure 3.1, the task is learned under a goal in a reinforcement learning framework, which as usual consists of states, actions, and rewards. The state vector contains sufficient information about the object for the current task, the set of actions contains the set of behaviors for learning affordances, and the reward is derived from a reward function related to 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 affordances and task learning are coupled in our architecture, the learned affordances are independent from task learning and can be reused in other tasks.

3.3.2 Tabular Affordance Learning

In this chapter, we only consider discrete object states, discrete robot actions and discrete effects. Subsequent chapters will discuss the continuous cases. In the discrete case, the knowledge of affordances is represented as an affordance table [T], consisting of triplets:

$$(o, a, e) \tag{3.3}$$

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 chapter, 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 may change. Although the current implementation is a simple affordance table, it allows further generalization with additional techniques.

3.3.3 Extended Learning Classifier System (XCS)

The task learning component needs to have efficient on-line learning performance for the robot to achieve its goal fast. This provides a good base for effective on-line affordance learning and the use during task learning. For this purpose, we choose an Extended Classifier System (XCS) for learning general rules in a reinforcement learning context.

Algorithm 1: On-line tabular affordance learning during a task.

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

XCS classifier system [85] is a rule-based system for reinforcement learning problems [56]. It can be regarded as a generalization of tabular Q-learning [90] by using a Genetic Algorithm (GA) [91] to aggregate similar states in the Q-table [92]. In XCS, the GA produces rules that make prediction of the reward when selecting the associated actions. After an action is performed and the actual reward is obtained, the prediction error is calculated and used to update all the relevant rules. While tabular Q-learning updates a single state-action pair, XCS updates multiple state-action pairs. For physical robot control, XCS has shown satisfactory results without careful tuning of parameters [93]. 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 [94]). A rule maps a condition and an action to a prediction, with an associated fitness as follows:

$$(condition, action) \rightarrow prediction : fitness \tag{3.4}$$

The rules can use real numbers, symbols or the classical ternary representation {0,1,#} to encode the conditions. The hash symbol # can be either 0 or 1. In this chapter, we choose the ternary representation. We use binary strings to encode the actions. The hash symbol # allows generalization and GA operations on the rule conditions with the same length.

For example, the rule

$$(0\#0\#11\#11, 001) \rightarrow 1000 : 0.59 \tag{3.5}$$

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 (3.5) is a value function, so the rules are value function fragments, and XCS uses GA techniques to generalize the value function [95].

In this chapter, the current state of the environment (see Figure 3.1) only considers the object, represented as:

$$s = (o, s_o) \tag{3.6}$$

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 assume that the state of the robot does not have to be included for the tasks considered in this chapter. 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:

$$[\mathrm{M}] = \bigcup_{i=1}^{M} [\mathrm{M}]_{a_i} \tag{3.7}$$

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

$$P(a_i) = \frac{\Sigma_{l \in [\mathrm{M}]_{a_i}} p_l F_l}{\Sigma_{l \in [\mathrm{M}]_{a_i}} F_l} \tag{3.8}$$

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 (3.8), an action a_j is selected by

$$a_j = \arg\max_{a_i} P(a_i) \tag{3.9}$$

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* [85], 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 Appendix A.

3.3.4 Affordance Use in XCS

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 our case, 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:

$$e_k \in \mathscr{E} \tag{3.10}$$

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 an XCS task learning system.

1: Initialize the affordance table [T] and the population of XCS rules [P] empty or load them from files.
2: Observe the current system state $s = (o, s_o)$.
3: Form [M] $\subset$ [P] whose rule conditions match s.
4: Calculate the prediction for each action using (3.8).
5: **if** o matches object o_k of (o_k, a_k, e_k) in [T] **then**
6: Filter a_k by its effect e_k using (3.10).
7: **end if**
8: Select an action a_j by using (3.9).
9: Apply a_j.
10: Observe the new system state $s' = (o, s'_o)$.
11: Update [T] by Algorithm 1 (Line 6 to Line 11).
12: Receive a reward r from the environment.
13: Form the action set $[M]_{a_j} \subset [M]$ which advocated a_j.
14: Call the *Credit Assignment Algorithm* to update $[M]_{a_j}$.
15: **if** Task is not terminated **then**
16: Goto Line 2.
17: **end if**

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 (3.10) 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 $[M]_{a_j} \subset [M]$ which advocated a_j is updated by the *Credit Assignment Algorithm* [92]. Refer to Appendix I for details.

3.4 Robot Platform

3.4.1 Hardware

All experiments in this dissertation were carried out with a humanoid robot NAO[1] (see Figure 3.2). The NAO is about 58 cm tall and weights about 4.3 kilograms.

Sensors Two cameras are located vertically on the forehead and at the mouth position, respectively. However, the cameras do not have an overlapping field of view and they cannot be used at the same time. Therefore, a single camera with a chosen resolution (e.g., 640×480) was used at one time. The sonars on the robot's chest can return distance readings (30 cm to 120 cm) to obstacles, and the tactile sensors on its head can be activated by a touch.

[1] http://www.aldebaran.com

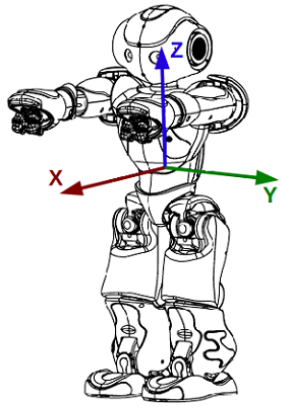


Figure 3.2: Body frame definition in the Cartesian space of a robot NAO.

Motors The academic version of NAO H25 has 25 degrees of freedom. The head has two motors (HeadYaw, HeadPitch), each arm has four motors (ShoulderPitch, ShoulderRoll, ElbowYaw, ElbowRoll), each hand has two motors (WristYaw, Hand), and each leg has five motors (HipRoll, HipPitch, KneePitch, AnklePitch, AnkleRoll) along with one HipYawPitch motor. In the experiments, different groups of motors were involved for different tasks. For example, a visual searching task involved the head motors, and a navigation task involved whole body motion (see Appendix I and II).

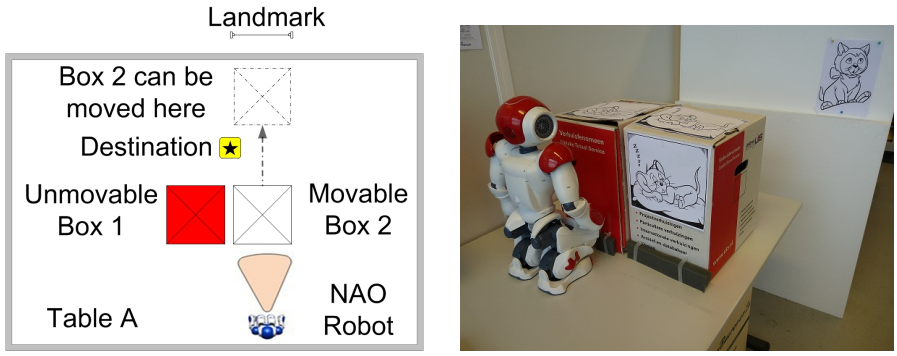
3.4.2 Software

The NAO ran an embedded Linux system which was remotely connected to a separate computer via Wi-Fi or Ethernet. The NAO collected raw sensory data and sent the data to the computer, where the data was processed and action commands were sent back to the NAO. This created an action perception loop that supported the robot to learn how to act based on sensory feedback. Machine learning algorithms were employed for processing the raw data as well as for making decisions on action commands[2]. To carry out an actual action command, default APIs were used. For example, "walking ahead" was defined as walking along the positive X-axis in the NAO space (see Figure 3.2). Similarly, the arms were controlled in the Cartesian space using available inverse kinematics.

3.5 Experiments and Results

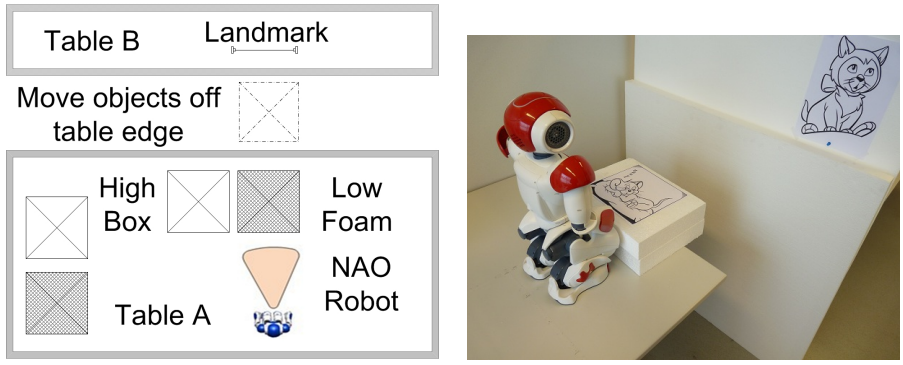
In the experiments, the movability of objects with different weights and sizes was investigated. The NAO used its whole body motion to push the object in front of it. It observed the effect, which was the location change of the object before and after the action (see Figures 3.3 and 3.4). Then, an affordance table was learned. With the affordance table, the

[2]Implementation details will be discussed in corresponding chapters.



(a) A navigation task. (b) NAO faced two boxes.

Figure 3.3: A humanoid robot NAO learned movability in a navigation task. Two boxes blocked the NAO's way and they were movable and not movable, respectively. The goal was to reach the destination as fast as possible. After several trials, the locations of the boxes were exchanged to show that the NAO were able to reuse the learned movability in a different environmental setting.



(a) A stacking task. (b) NAO faced a piece of foam.

Figure 3.4: A humanoid robot NAO learned movability in a stacking task. The NAO stood on the table edge and chose different pushing poses for a high box or a piece of low foam. The goal was to push them off the table to make a stack as high as the table. The height of a high box was 36 cm and the height of a low foam was 12 cm.

robot was expected to operate more effectively and avoid making mistakes repeatedly in a task, e.g., pushing the same unmovable box even though it had failed for several times.

Sequential goal-directed tasks were designed to test the proposed model on the NAO. We assumed that the robot was 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 had a repertoire of preprogrammed behaviors to interact with the environment. We used XCSlib [96] for task learning.

3.5.1 Environment and Tasks

Given a map and a landmark in the world space, the NAO was able to localize itself and guaranteed its safety on a flat table. A marker was used on each object for object recognition and for measuring the relative object location and distance to the NAO.

3.5.2 Movability

As defined in equations (3.2) and (3.3), movability was learned automatically in a task whenever the robot tried to interact with objects by pushing. In our case, pushing the object o with the action a would make it move a distance of d_j, which was thresholded to be 1 or 0, meaning "moved" or "unmoved" respectively.

3.5.3 Sensory Input

The forehead camera was used as the main sensory input (640×480 resolution) and the sonars on its chest confirmed there was an object in front of the robot. The NAO localized itself by matching SIFT [88] features of the landmark with known 3D coordinate (x_l, y_l, z_l) in the world space. This provided the NAO its camera location (x_r, y_r, z_r) in the world space [68]. When the marker on the object was detected, the NAO calculated its relative location to the object as $(x_{r2o}, y_{r2o}, z_{r2o})$. The object's location (x_o, y_o, z_o) was then obtained, which was discretized as a 4-bit binary string S_{xyz} to represent s_o. Meanwhile, the height of the object h_o was obtained by:

$$h_o = h_{r2t} - |z_{r2o}| \tag{3.11}$$

where h_{r2t} was assumed known as the height of the NAO's camera relative to the table surface in its normal standing pose, and $|z_{r2o}|$ was the approximate vertical distance from the camera to the object. Then, h_o was represented by o_{height} where 1 meant the object was high and 0 meant low. The colors of the boxes were represented by a 3-bit binary string o_{color}. In total, s in equation (3.12) was a 8-bit string:

$$s = (o, s_{xyz}) \tag{3.12}$$

in which

$$o = (o_{height}, o_{color}) \tag{3.13}$$

3.5.4 Actions

The NAO was provided with eight actions represented as a 3-bit binary string shown in Table 3.1. In the navigation task, the first four actions were used. The NAO moved a distance of d_{walk} left, right, ahead, or pushed when walking ahead. In the stacking task, the NAO stood on the table edge and chose the latter four actions in Table 3.1. To simplify the process of finding and taking an object to the table edge, the NAO chose an object by requesting the human operator with an "text-to-speech" command. Then, the NAO confirmed that the requested object was in front before trying to push it in two different poses, standing or squatting.

Table 3.1: 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)

3.5.5 Action Filter

The action filter in equation (3.10) filtered those actions which were predicted to have the "unmoved" effect on the object, e.g., pushing an unmovable object or pushing a movable low object in a wrong pose.

3.5.6 Reward Function

After the action a_j was performed at the previous time step, the NAO received an immediate reward of r in the current time step.

In the navigation task, if its current distance to the destination d was smaller than a threshold d_{dst}, a final reward of 1000 was given. If its location change Δd was less than d_{move}, it was punished by -100. This is described by the following reward function:

$$r = \begin{cases} 1000 & \text{if} \quad d < d_{dst} \\ -100 & \text{if} \quad \Delta d < d_{move} \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

In the stacking task, if the stack was as high as the table, the goal was achieved and a final reward of 1000 was given. Whenever the robot chose an object to push, it received a punishment of -100. Otherwise, $r = 0$. This is described in equation (3.15):

$$r = \begin{cases} 1000 & \text{if} \quad \text{stacked} \\ -100 & \text{if} \quad a_j = \text{choose (object)} \\ 0 & \text{otherwise} \end{cases} \tag{3.15}$$

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

$$payoff = \Sigma r \tag{3.16}$$

3.5.7 Experimental Setting

In both tasks, an experiment consisted of alternated explore and exploit trials. They used the same parameter setting of XCS (Refer to Appendix A for the details). The number of action steps in one trial was limited to the maximum of $n_{step} = 10$. If the goal was achieved or n_{step} actions were performed, the trial was terminated and the environmental setting was initialized as before.

During the first 10 explore/exploit trials in the navigation task, the unmovable box was put on the left side of the movable one (see Figure 3.3(b)). Then, the boxes were exchanged and the remaining 10 trials were performed. In this task, the table area was $0.75\,\text{m} \times 0.6\,\text{m}$, and $d_{walk} = 0.15\,\text{m}$. The NAO was allowed to start from anywhere on one side of the area aiming for the destination which was 0.4 m away on the other side of the area. The standard XCS and the affordance-based XCS were used in this task to compared the required number of actions for achieving the task goal.

A video is available at http://youtu.be/1yH6TwWtByU.

3.5.8 Results

In the sequel, XCS rules will be given to illustrate what have been learned in the navigation task. Then, we show the advantage of our cognitive architecture that learns and uses affordances to aid task learning.

XCS Rules

Table 3.2 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 (3.12) and (3.13). Take rule 19 for example, when facing the object "10#0" (high and probably a white box) which was in the area "#110" (the middle part of the table, either left or right), if the NAO took the action "010" (push 0.15 m ahead), the predicted reward would be 1000 (goal achieved), and the error of this prediction was 1.0. This rule had a fitness of 0.59, and the average action set size was 3.3. It had been activated for 4 times and the numerosity of this rule was three.

In the first 10 trials, if the NAO faced the object "1000" in the area "1000", it was most likely to choose rule 7, 9 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. In this area, old specific rules like 7 and 9 did not match any more, but more general rules like 2 and 19 still matched. Besides, new specific rules like 45, 94, 106,146 also suggested the pushing action.

Table 3.2: XCS Rules learned in the navigation task.

ID	Condition	Act.	Pred.	Error	Fit.	Size	E.	N
2	1####000	010	138.1	137.9	0.15	6.9	10	3
5	###01000	011	-94.0	0.5	0.30	2.5	2	2
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
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
19	10#0#110	010	1000	1.0	0.59	3.3	4	3
21	100#1010	011	-33.6	29.7	0.17	2.5	2	3
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
26	#1#000#0	011	-111.9	9.5	0.37	1.5	2	2
30	#1100#00	000	-93.0	1.0	0.37	3.0	2	3
36	#11#0010	001	-93.0	1.0	0.37	1.0	1	1
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
60	11#1####	001	10.0	1.0	0.01	1.0	0	1
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
137	1#1#####	000	109.9	24.8	0.15	7.3	3	1
146	1#####00	010	672.1	163.0	0.14	8.7	3	1

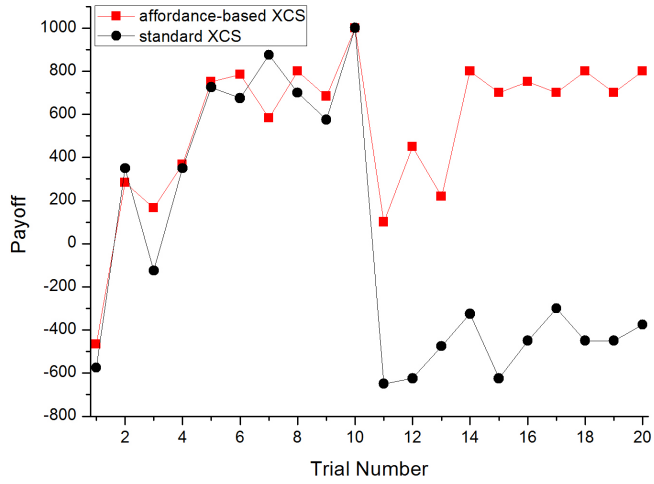


Figure 3.5: Average payoff in the navigation task.

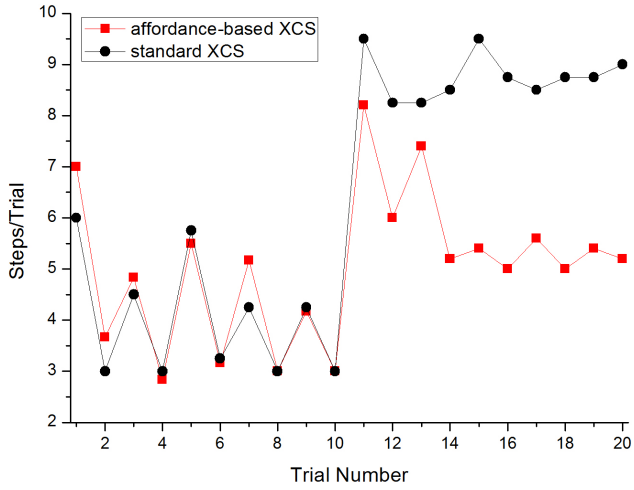


Figure 3.6: Average number of steps in the navigation task.

Affordance-Based XCS versus Standard XCS

As mentioned in Section 3.5.7, odd trials were always exploratory while even trials were exploiting. Figures 3.5 and 3.6 show the average payoff defined in equation (3.16) and the average number of action steps in each trial. The maximal payoff is 1000, and the maximal number of action steps is 10. The result was averaged over 5 experiments.

In the first 10 trials, both affordance-based XCS and standard XCS were able to solve the problem from the 4th trial. It was not difficult for the NAO to find the two-step solution to push the movable box. In some cases, the NAO reached the boundaries of the table and the trial was terminated. When the robot ran into the unmovable box, affordance-based XCS performed better than standard XCS because the NAO would avoid pushing the unmovable again. However, on average, there were not much differences in payoff or number of steps because of some lucky trials in which the NAO did not run into the unmovable 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. This was due to that overly general rules were learned in the first 10 trials which suggested the NAO to push the object as before. 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.

We can compare the success chance between the two approaches in the second 10 trials. In this case, the NAO started from facing the unmovable box and needed to move 2 steps left (would otherwise received a punishment of -200) and push the movable box for 3 steps

forward. Without considering the influence of XCS rules, the chance of moving 2 steps left would be 1/16 for standard XCS and 1/4 for affordance-based XCS after having learned the movability.

Learned Movability

Table 3.3 shows the learned movability of the objects.

Table 3.3: 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

Similar to the XCS rules, the triplets with larger ID were generated later. Actually, we can find the relations between the triplets in Table 3.3 and the rules in Table 3.2. For example, triplet 1 in Table 3.3 is related to rule 39 in Table 3.2. They suggest that action "010" did not move the object "1111" and a reward of −93 is predicted, respectively.

In the navigation task, Table 3.3 filtered unwanted actions by triplets 1, 2 and 4 which meant that the NAO pushed the unmovable box or part of it with no effect. In contrast, triplet 3 meant that the NAO pushed the movable box forward. 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. Then, the NAO tried to squat first before pushing the low object. Finally, the NAO succeeded in learning the policy to choose the high box rather than the low foam. Because choosing a high box required less actions to achieve the stacking goal which would result in a higher accumulated reward.

3.6 Conclusions and Open Issues

In this chapter, we have proposed an architecture to learn and use affordances in goal-directed tasks. We have integrated task learning, affordance learning and affordance use in a general framework, in which affordances can be obtained and used automatically during on-line task learning. In the architecture, the task is learned given a goal, and affordances are learned by means of updating an affordance table, while affordances are used to select actions for task learning.

An experimental evaluation of the proposed architecture has been carried out. Reinforcement learning has been chosen as the approach for on-line task learning, specifically the XCS classifier system has been used. The affordance-based XCS has shown a good performance in real-world tasks. Compared with standard XCS, the affordance-based XCS shows various improvements. In the navigation task, changing the locations of movable

and unmovable boxes has resulted in failures for standard XCS that suggested to keep pushing the unmovable box, while the affordance-based XCS has succeeded to achieve the goal by pushing the movable box. This indicates that using affordances is more efficient for avoiding the selection of wrong actions that would not contribute to achieving the goal. Selecting the same actions repeatedly is common for RL systems such as XCS even though the actions are undesired. It usually takes some time for RL systems to change the action selection policy until rewards are given. In contrast, the additional knowledge of affordances directly provides suggestions on action selection, without the need to wait for rewards. For this purpose, designing a proper action filter is the key to decrease the number of candidate actions. The action filter in this chapter filters out the actions that result in no object displacement in the navigation task. However, it is difficult to design a generic action filter for a wide range of goal-directed tasks. It is interesting to investigate how to design action filters based on given goals. One possible approach is using high-level action rules as will be discussed in Chapter 6.

In the proposed architecture, only the task-relevant affordances are learned and used. The process of data collection for affordance learning is on-line during task learning whenever an action is performed on an object. The learned affordances are also simultaneously used to speed up the task learning. In contrast, the two-staged developmental framework [33] requires a robot to interact with objects with all its actions (called "motor babbling" [43, 28, 37]) to learn affordances before it can use the affordances to achieve a goal. In such a case, it is not known a priori what the goal is going to be, therefore all kinds of affordances are initially learned. As a result, the robot can spend much time learning affordances which might be irrelevant to the goal. This indicates that our approach is more efficient for solving a goal-directed task than the two-staged approach.

This chapter has used a binary representation of object features, and the affordances of known objects are used for action selection. The affordance table keeps the latest data of learned triplets, and the latest data replaces the old data if there is a conflict. This means that our approach is able to adapt to dynamic environments in which the affordances may change. Maintaining the long-term statistics of the affordances is also useful for handling uncertainty, e.g., by using probabilistic models [10]. A direct extension of the current approach would be to replace the binary representation of objects with a ternary representation as used in the XCS rules (see equation(3.4)). As a result, the affordances could generalize in the same way as the XCS rules generalize over state spaces. However, the generalization of affordances comes at the price of computational complexity compared to the simple affordance table.

While the proposed architecture has been shown effective for discrete object states and actions, in principle there is no obstacle to handling continuous spaces. In particular, the continuous version of object representation (see section 2.2.3) could be used to capture the distinctive object features, function approximators could be used to learn affordances, and a continuous version of task learning algorithm could be used. However, continuous actions would be difficult to handle in the proposed architecture, because there are infinitely many data points in continuous spaces. We will further address this issue in Chapter 4.

Chapter Four

Active Affordance Learning in Continuous State and Action Spaces

This chapter introduces an active affordance learning architecture that works in continuous spaces. We use function approximation to model the relations between object states, robot actions and effects, which allows generalization in continuous spaces. While learning affordance models, a continuous version of the actor-critic reinforcement learning module enables the update of exploration policies as well. We address the challenge of active exploration in continuous action spaces, unlike previous approaches from the literature, which typically use a set of discrete actions to perform random exploratory actions. Also, our architecture allows a robot to decide by itself when the learning can terminate, based on the convergence of the actor-critic module. The learned affordance models can be reused to acquire a range of manipulation skills in a bottom-up manner. Robot experiments have been carried out to solve garbage can manipulation tasks using a one dimensional state space and three dimensional action space. In these experiments, the robot efficiently learns affordance models and reliably makes use of them to open and close garbage cans of different types.

4.1 Introduction

In the previous chapter, we proposed a cognitive architecture that supports affordance learning and use in discrete state and action spaces. The present chapter concerns continuous spaces without manual discretization of the state space or manual tuning of the action parameters. The main motivation is that the discrete representation of object states or robot actions is not always effective in solving real-world tasks. In such tasks, not only object states have to be configured in the continuous world space, but also successful manipulation requires the corresponding action parameters to be learned by the robot. For example, garbage cans come with different designs, shapes and sizes (see Figure 4.1). When opening the garbage can in Figure 4.1(b), the robot needs to learn where on the lid and in which direction a force should be exerted, as well as how much the lid should be opened to dispose a given item. It is not intuitive to provide by human a representation of every household object along with preprogrammed motor skills for handling it. Therefore, it is necessary to develop a robot learning approach for object manipulation in continuous spaces.

(a) "lift" lid (b) "push" lid (c) swing lid (d) pedal

Figure 4.1: Examples of garbage cans with movable lids. (a) a bucket handle affords grasping and lifting, (b) a lid affords pushing forward, or (c) downward, and (d) a pedal affords pressing it downward, typically by a foot.

There has been much research on robot skill learning via human demonstration [47, 97]. Complex robot motions can be learned by mixing basic motor primitives [98, 99], which is quite effective for object manipulation with human-provided training samples. Robots are able to learn skills such as how to flip a pan cake [100], throw a dart [101], play table tennis [102], serve tea [49], etc. However, these approaches require significant human efforts to provide a library of high-quality motor primitives. Such human guidance is not always available or is too expensive to obtain. Moreover, these robot skills are targeted at specific tasks for handling specific objects. The focus is on the generation of robot motions, without much consideration of object representation. Consequently, the learned skills can hardly be transferred for handling novel objects. In general, it is still an open challenge for autonomous robot learning of object manipulation skills.

This chapter proposes an active robot learning approach without human demonstration. Through intrinsically motivated action exploration, the robot can actively collect training data by itself for efficient affordance learning and skill acquisition in continuous state and action spaces. In psychology, intrinsic motivation refers to the doing of an activity for the enjoyment of the activity itself, while extrinsic motivation, in contrast, refers to the doing of an activity in order to attain some separable outcome, e.g., external rewards [103]. Intrinsically motivated activities include playing games for fun, learning because of curiosity, etc. They favor the development of broad competence rather than being driven by specific task goals. In our case, the intrinsic motivation is to learn an accurate affordance model that predicts action effects on objects. In other words, the robot aims to reduce the prediction errors of the affordance model. This drives the robot to actively explore uncertain areas in the state and action spaces in which the prediction errors are high. The active exploration process will be terminated when the learned model makes accurate enough predictions. Thereafter, the learned model is used to select actions to achieve goals, during which a range of manipulation skills are acquired autonomously.

The remainder of this chapter is structured as follows. Section 4.2 gives a brief overview of the literature related to our approach. Section 4.3 proposes the robot learning architecture that supports active affordance learning and manipulation skill acquisition. Section 4.4 describes the details of active affordance learning in continuous state and action spaces.

Section 4.5 discusses how manipulation skills can be acquired by using the learned affordances. Section 4.6 reports the results of real robot experiments, in which our learning architecture is used for manipulating garbage cans. Section 4.7 concludes the chapter.

4.2 Related Work

Affordance learning in continuous action spaces has been considered in [43, 10, 28, 38, 37]. However, the affordance learning conditions have been strongly controlled by programmers, and this restricts the autonomy of the robot. Usually, the amount of needed data is decided before the robot actually starts learning. In addition, the data is collected through random motor babbling [19]. This is not efficient in continuous state and action spaces that contain infinitely many data points.

In machine learning, active learning is a technique that allows active selection of training data for learning prediction models [104]. In robotics, active learning has been applied for efficient acquisition of the training data for learning knowledge and skills. For example, a robot actively generates uncertain situations and queries a human teacher [105, 106] to reduce the amount of training data for learning symbolic concepts. In [107], the robot also actively asks human about how to handle novel situations to speed up the learning of goal-oriented hierarchical tasks. However, in this chapter, we do not assume that interactive human guidance is available. In other words, human intervention is not allowed when the robot starts learning.

Without human guidance, active learning can be driven by intrinsic reward signals in the framework of intrinsically motivated reinforcement learning [71, 108]. Such reward signals include curiosity, surprise, fear, etc. [109, 70, 110]. Heuristics typically direct active exploration towards the regions where uncertainty or prediction errors are maximal [111]. However, problem spaces can be too difficult to learn or are unlearnable by the robot, which means the prediction errors are always high[1]. In this case, the change of prediction error is used instead of the prediction error as the intrinsic reward to optimize the learning progress [112, 113]. Then, the robot intends to explore the motor spaces with medium-level prediction errors. In this chapter, however, we consider affordance learning with household objects that are assumed manipulable and learnable by the robot. Therefore, we take the approach of using prediction errors as the reward signal. Nevertheless, we do not exclude other approaches for providing reward signals. Our active learning architecture is flexible enough to handle a variety of reward signals.

Another relevant approach [114] proposes active learning of controllable environmental contexts for object manipulation. However, the motor skills are preprogrammed and high-level control programs are given. In contrast, we assume no preprogrammed manipulation skills or high-level control programs. In our approach, a range of manipulation skills are acquired in a bottom-up manner without manual discretization of the continuous state space or robot action space.

The main contribution of this chapter is that we propose an architecture that facilitates active affordance learning and manipulation skill acquisition in continuous spaces. We choose the framework of intrinsically motivated reinforcement learning for active affor-

[1] Refer to [112, 113] for such examples.

dance learning. Specifically, we use the actor-critic reinforcement learning (RL) architecture [56] to learn action exploration policies. Forward models [115] are learned through function approximation and used to predict action effects in continuous state and action spaces. The prediction error is not only a means to update the forward models, but it also serves as the intrinsic reward signal to update the critic and the actor. Then, the learned forward models are reused to select goal-directed actions, during which a range of manipulation skills are acquired in a bottom-up manner. In this way, the robot learns to handle objects through its own sensory and motor experience.

4.3 Active Learning Architecture

The overall architecture we propose for active affordance learning and manipulation skill acquisition is illustrated in Figure 4.2. It consists of three components: affordance learning, active exploration, and model exploitation.

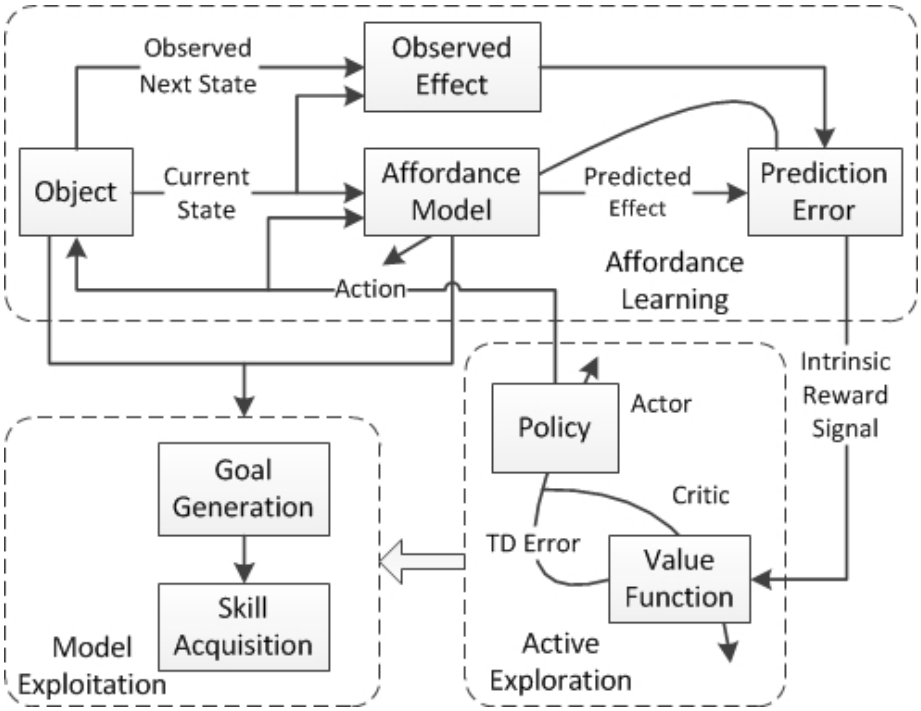


Figure 4.2: An architecture of active affordance learning and goal-directed skill acquisition.

Affordance Learning Component

In the affordance learning component, we define an affordance model that associates the three elements of *Object*, *Action*, and *Effect*, the same as in **Definition** 3.1. However, unlike

the previous chapter that considers discrete object states and robot actions, this chapter handles continuous spaces. Due to that there are infinitely many data points in continuous spaces, the memory for storing the triplets in equation (3.3) would increase dramatically as the robot collects more and more data after interacting with objects. Therefore, we take the approach of function approximation to represent an affordance model in continuous spaces. As a result, the affordance model can generalize over the entire state and action spaces based on the collected data.

The affordance model is represented as a forward model [115] that produces a prediction of an effect, based on the state of object and the selected action. An object is represented by a bounding box perceptual proxy[2] (see Figure **??**). In this way, we can obtain the state of object as the size and location of the bounding box. Consequently, affordance learning is to learn a forward model which is associated with a perceptual proxy of an object. The learning of the forward model is based on the current object state, the robot action performed, and the model prediction error, i.e., the difference between the predicted effect and the observed effect. Supervised learning algorithms can be used for learning forward models [116]. Specifically, we choose a feed-forward neural network as the model and use back-propagation to update the model parameters based on the prediction error. For example, a robot trains the feed-forward neural network to predict object displacement as a result of its arm movement. Object state is represented as the current location of the object in the world space, e.g., as done in the experiments of chapter 3. Whenever a pushing action is applied to the object, the neural network makes a prediction about the consequent object displacement. After the next object state is observed, the actual displacement is measured by comparing the object states before and after the action, i.e., by calculating the location change in the world space. Then, the prediction error of the displacement is calculated and used to update the parameters of the neural network.

Active Exploration Component

In the active exploration component, the output is an action for the affordance learning component. The action is not only to be performed on the object, but also sent to the affordance model for effect prediction. As a baseline, a random action can be selected. However, random exploration is not efficient in a continuous action space. We propose to use an actor-critic reinforcement learning (RL) module for active selection of exploratory actions. In the actor-critic RL architecture, the actor plays the key role that determines the new samples for learning the affordance model. The critic learns to predict the value of each state and computes the Temporal Difference (TD) error [56], which is used by the actor to output optimal actions that will maximize the accumulated future rewards.

The reward signal comes from the affordance learning module. This is a kind of intrinsically motivated RL that the reward function is not designed for specific goals such as navigating to a destination. In this chapter, the prediction error of the affordance model serves as the reward signal. The intrinsic motivation of the robot is to make better prediction of action effects using the affordance model. The underlying heuristic is straightforward: the affordance model is maximally corrected when the sampled state and action spaces have the highest prediction error. In other words, sampling in state and action spaces with higher

[2]Other options include centroid perceptual proxy and ellipse perceptual proxy. Refer to [54] for details.

prediction error is more rewarding than sampling in the already well-predicted area. As a result, the prediction error of the affordance model is expected to decrease and become stable after sufficient data have been collected. Finally, active exploration can terminate when the TD errors converge, which means there is not much change in the actor-critic module.

We note that the two components of affordance learning and active exploration interact with each other in a loop. An affordance model is learned simultaneously with an exploration policy. The affordance learning is carried out in an active manner in the sense that the robot actively decides by itself what actions are to be performed to collect data for learning the affordance model. We will discuss the details in section 4.4.

Model Exploitation Component

In the model exploitation component, the learned affordance model is used to acquire manipulation skills. In order to acquire useful skills, the learned models are used to solve goal-directed tasks. We propose an approach to self-generate goals and select a sequence of actions to achieve the goals.

The generation of a goal can be done in several ways. A straightforward approach is to program a goal manually as a specific condition, e.g., having arrived at the destination in a navigation task (see section 3.5.6). However, this requires that the human programmer has the complete knowledge of the task environment. Another interactive approach is to demonstrate the goal to the robot. By observing human demonstration, a robot can emulate a goal in the object state space [33], or interpret a goal in the effect space [10].

In this chapter, we take a more autonomous approach involving less human supervision. Based on the observation of the current object state and the learned forward model, the robot samples its action space and estimates the maximal possible change of the object state, i.e., the effect on the object. This effect is selected as the current goal, and the corresponding action that is expected to achieve the goal is the desired manipulation skill. For example, pushing a door in different directions results in different door opening sizes. After having tried various pushing directions and learned a forward model, the robot generates a goal to maximize the door opening size by finding the most effective pushing direction.

By achieving this goal, a skill is acquired. The skill consists of three elements: 1) the initial object state, 2) the selected action to achieve the goal, and 3) the actual effect of the action. The skill acquisition process continues until a termination condition is satisfied, i.e., the effect is not changed anymore. A range of skills can be acquired by exploiting the learned forward model in various configurations of initial states. In this way, the robot can develop object manipulation skills autonomously when no task is specified by a human. In the case of garbage can manipulation, a robot can learn to open the garbage can when it is closed or half-open, close it when it is open, or move it around to desired locations. We will discuss the details in section 4.5.

4.4 Active Affordance Learning in Continuous Spaces

We now discuss in detail the affordance learning component and the active exploration component in Figure 4.2.

4.4.1 Affordance Model in Continuous Spaces

We follow the definition of affordance model in Section 3.2. Unlike in the previous chapter, we instantiate the three elements *Object*, *Action*, and *Effect* in continuous spaces, and introduce the forward models that are used to model affordances in continuous spaces.

Perception of object and parts

We consider household objects that have movable parts, e.g., refrigerators, microwave ovens, kitchen cupboards, drawers, etc. We assume that based on known markers and color segmentation[3]. The robot then recognizes a household object as a combination of all observed parts.

Denote by o a household object represented by the bounding box perceptual proxy. We still use $s_o \in \mathcal{S}_{obj}$ to denote the state of the object o, where $\mathcal{S}_{obj}$ is a predefined continuous attribute space. Denote by Ψ the set of all known object parts (body, lid, handle, pedal, etc.). As not all objects necessarily contain the same parts, denote by $\Psi_o \subseteq \Psi$ the set of parts that the object o is composed of. We note that $\mathcal{S}_{obj}$ usually describes an object as one whole piece, e.g., its location and its center of mass in the world space. $\mathcal{S}_{obj}$ may also include the states of individual object parts if necessary.

Continuous Action Spaces

Robot actions can be defined in the constrained joint space as well as in the Cartesian space. In this chapter, we control a robot arm in 3D Cartesian space using inverse kinematics. This does not exclude other continuous action representations in our active learning architecture.

Denote by $s_r = (x, y, z)^T \in \mathbb{R}^3$ the current state of a robot's end-effector in the 3D space. Denote by $a = (\Delta x, \Delta y, \Delta z)^T \in \mathcal{A} \subset \mathbb{R}^3$ a bounded action that changes the position of the end-effector. In our case, the robot interacts with only one part of the object using one end-effector at a time. The robot always approaches the vicinity of the chosen object part before interacting with it. The reaching and grasping behaviors are assumed available in the robot's motor skill repertoire.

Continuous Effect Spaces

The effect of action a on object o is denoted by $e_o \in \mathcal{E}_o$, where $\mathcal{E}_o$ is a continuous effect space. An effect is measured by equation (3.2).

Forward models

A forward model is an internal model that produces a predicted output based on a given input [115]. In our case, the input is the current object state and the applied action, and the output is the predicted effect. Then, object affordances are encoded in the following forward models $\mathcal{F}_\psi$:

$$e_o = \mathcal{F}_\psi(s_o, a, w) \text{ for } \psi \in \Psi_o \tag{4.1}$$

[3]Refer to [117], the robot can identify object parts using an RGB camera for a sophisticated computer vision method to recognize object parts with a RGB-D camera. As our focus is on active learning, such a method is beyond the scope of this chapter.

where ψ indicates that the robot interacts with a specific part $\psi \in \Psi_o$. In our case, $\mathcal{F}_\psi$ is a feed-forward neural network and w is the weight vector.

4.4.2 Learning Forward Models

In continuous state and action spaces, affordance learning is translated to learn the forward model (4.1) by updating the model parameters based on prediction errors. We use an on-line version of neural networks. Denote by $(s_o^k, a^k, e_o^k), k \in \mathbb{N}$ the collected data after applying an action a^k, where s_o^k and e_o^k are the corresponding object state and consequent effect. The decision of data sampling and its termination will be discussed in Section 4.4.3.

When learning a forward model $\mathcal{F}_\psi$, denote by $\hat{e}_o^k$ the predicted effect of a^k in the state s_o^k, i.e.,

$$\hat{e}_o^k = \mathcal{F}_\psi(s_o^k, a^k, w^k) \tag{4.2}$$

where w^k is the current weight vector. The prediction error η_k is obtained as follows:

$$\eta_k = e_o^k - \hat{e}_o^k \tag{4.3}$$

where $e_o^k = m(s_o^k, s_o^{k+1})$ is obtained from equation (3.2). Then, the new model parameter w^{k+1} is updated as follows:

$$w^{k+1} = w^k + \alpha\, \eta_k \nabla \mathcal{F}_\psi(s_o^k, a^k, w^k) \tag{4.4}$$

where $0 \leq \alpha \leq 1$ is the step size parameter, and $\nabla \mathcal{F}_\psi$ is the gradient of the output of the network to the weight vector.

4.4.3 Active Learning with Intrinsic Motivation

The goal of active affordance learning is to autonomously learn the relations between objects, actions and effects in an efficient manner. Meanwhile, the policy of selecting exploratory actions should also be learned to optimize the affordance learning process. A baseline to be compared with is the random action selection policy.

In order to learn the exploration policy, we integrate an RL component in the affordance learning loop (see Figure 4.2). A conventional RL scheme requires the definition of a reward function to develop goal-directed exploration behaviors for a specific goal. In our architecture, the reward signal is generated intrinsically by using the prediction error of a forward model, whose maximization is expected to result in an optimal action selection policy.

Specifically, we have chosen Continuous Actor-Critic Learning Automation (CACLA) because it has been proved to have good performance for RL problems in continuous action spaces [118]. Like other actor-critic algorithms, CACLA is based on the simultaneous on-line approximation of two structures, the actor and the critic. The actor corresponds to an action selection policy, mapping states to actions in a probabilistic manner. The critic corresponds to a value function, mapping states to expected cumulative future reward.

An actor is represented as a function approximator Act_k that approximates the function $Act^* : S \rightarrow \mathcal{A}$, where $Act^*(s_o^k)$ denotes the optimal action for state s_o^k. The critic is represented as a function approximator V_k that approximates a state value function $V : S \rightarrow \mathbb{R}$

which stores the expected sum of discounted rewards for states. The strategy of active exploration is learned as follows.

During exploration, an action a^k is sampled from the Gaussian probability density function $G(x, \mu, \sigma)$ centered around the output of the current actor $Act_k(s_o^k)$:

$$G(x, Act_k(s_o^k), \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-Act_k(s_o^k))^2/2\sigma^2} \tag{4.5}$$

where σ is an exploration parameter. If $\mathcal{A}$ is more than one dimensional, σ could be chosen separately for each dimension.

After action a^k is applied on the object, the new state s_o^{k+1} is observed, and the actual effect is compared with the predicted effect to get the prediction error by equation (4.3). The current reward is given as the absolute value of this prediction error:

$$r = |\eta_k| \tag{4.6}$$

Then, the TD error [56] is obtained as follows:

$$\delta^k = r + \gamma V_k(s_o^{k+1}) - V_k(s_o^k) \tag{4.7}$$

where $0 \leq \gamma < 1$ is the discount factor.

The current actor Act_k is updated only if $\delta^k > 0$, which means that the performed action a^k is better than expected and should therefore be reinforced. The actor Act_k is then updated towards this action:

$$Act_{k+1}(s_o^k) = Act_k(s_o^k) + \zeta(a^k - Act_k(s_o^k)) \tag{4.8}$$

where $0 \leq \zeta \leq 1$ is a step size parameter.

The critic is always updated with the TD error:

$$V_{k+1}(s_o^k) = V_k(s_o^k) + \beta\, \delta^k \tag{4.9}$$

where $0 \leq \beta \leq 1$ is also a step size parameter.

The action exploration process terminates when the RL almost stops, i.e., there is not much change in the actor-critic RL component. This is measured by the convergence of the TD error, when the following condition is satisfied:

$$|\bar{\delta}^{k+1} - \bar{\delta}^k| < \epsilon \tag{4.10}$$

where ϵ is a small positive threshold, and

$$\bar{\delta}^k = \frac{1}{N} \Sigma_{i=k-N+1}^{k} |\delta^i| \tag{4.11}$$

is the averaged absolute TD error of recent N time steps.

The whole loop of active affordance learning is summarized in Algorithm 3. In case of endless exploration, the loop terminates anyway after a maximal number N_s of time steps.

Algorithm 3: Active affordance learning in continuous state and action spaces.

Require: An object o; Maximal time steps N_s;
for all $\psi \in \Psi_o$ **do**
  Initialize $k = 1$;
  while $k \leq N_s$ and (4.10) is not satisfied **do**
    Observe the object state s_o^k;
    Select an exploratory action a^k using (4.5);
    Predict the action effect $\hat{e}_o^k$ using (4.2);
    Apply a^k and observe the resulted object state s_o^{k+1};
    Calculate the prediction error η_k using (4.3);
    Update the parameter w^k of $\mathcal{F}_\psi$ using (4.4);
    Calculate the intrinsic reward r using (4.6);
    Calculate the TD error δ^k using (4.7);
    if $\delta^k > 0$ **then**
      Update Act_k using (4.8);
    end if
    Update the critic V_k using (4.9);
    $k \leftarrow k + 1$;
  end while
end for

4.5 Goal Generation and Skill Acquisition

A goal is generated in the continuous effect space $\mathcal{E}_o$. Denote by s_o the current object state, e.g., the opened width of a door. In each dimension of the effect space $\mathcal{E}_o$, the robot rehearses internally by sampling actions in the continuous space $\mathcal{A}$ and making prediction of corresponding effects. Then, it finds the maximal effect it can make and defines this effect as the current goal e_o^g, e.g., maximal door opening. In other words, e_o^g is calculated as follows:

$$e_o^g = \max_{a \in A_M} \mathcal{F}_\psi(s_o, a, w),\ \psi \in \Psi_o \tag{4.12}$$

where $A_M \subset \mathcal{A}$ is a selected set of M actions, e.g., evenly sampled in $\mathcal{A}$. Then, the robot selects and performs the corresponding action a^* that is predicted to result in the effect e_o^g:

$$a^* = \arg\max_{a \in A_M} \mathcal{F}_\psi(s_o, a, w),\ \psi \in \Psi_o \tag{4.13}$$

The consequent object state s'_o is observed, and the actual effect e_o is calculated using equation (3.2).

We represent the acquired manipulation skill as follows:

$$SK^* = \{s_o, a^*, e_o\} \tag{4.14}$$

The process of maximizing effects continues until no more effect is observed, which is measured by the following termination condition:

$$|e_o| < \tau \tag{4.15}$$

where τ is a small positive real number.

Then, the skill acquisition process continues by minimizing effects in the current dimension of $\mathcal{E}_o$, e.g., to close the door. Accordingly, equation (4.12) is changed into

$$e_o^g = \min_{a \in A_M} \mathcal{F}_\psi(s_o, a, w),\ \psi \in \Psi_o \tag{4.16}$$

and the corresponding action a_* is selected by

$$a_* = \arg\min_{a \in A_M} \mathcal{F}_\psi(s_o, a, w),\ \psi \in \Psi_o \tag{4.17}$$

We still use equation (4.15) as the termination condition.

We represent this new skill as follows:

$$SK_* = \{s_o, a_*, e_o\} \tag{4.18}$$

In this way, SK^* and SK_* can be acquired as paired skills, e.g., open the door and close the door. These skills can be reused directly when a specific task goal is given later on.

4.6 Experiments

We used a humanoid robot NAO and two garbage cans to test our approach (see Figure 4.3).

4.6.1 Task Setting

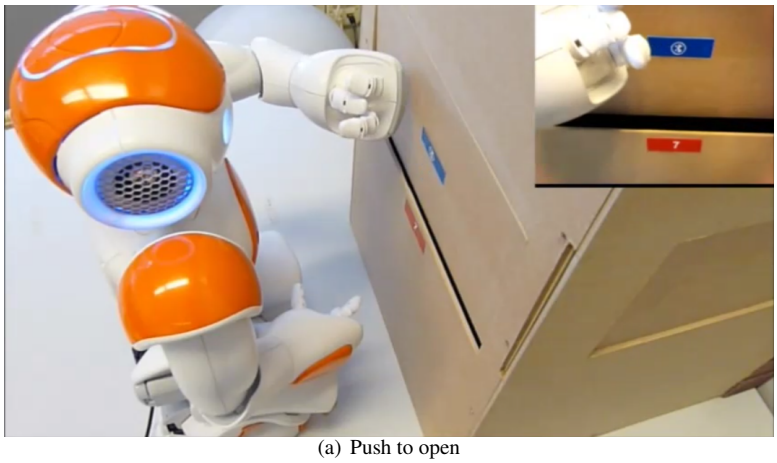
In our experiment, the garbage cans were presented to NAO separately. One had a pushable lid (Figure 4.3(a)), and the other had a pullable handle (Figure 4.3(b)). In each learning task, a garbage can was positioned approximately 10 to 12 cm in front of NAO and the area to be explored was about 25 to 45 cm high. These values agreed with the capabilities of NAO according to its height and the length of its arms. Only the left arm of NAO was used to interact with the garbage cans. The garbage cans were reachable and manipulatable by NAO.

The bottom camera on NAO's head was used as the main sensory input, with a resolution of $W \times H$ (e.g., 320×240). For each garbage can, the same blue marker ($5\,\text{cm} \times 2\,\text{cm}$) was used for the recognition of lid (with a NAO marker at its center), and a green marker for the recognition of the handle ($10\,\text{cm} \times 1\,\text{cm}$), if there was one. The markers were recognized based on color segmentation. As a result, the set of object parts was $\Psi = \{\psi_l, \psi_h\}$ where ψ_l denoted a lid, and ψ_h denoted a handle. Each $\psi \in \Psi$ was located by a bounding box.

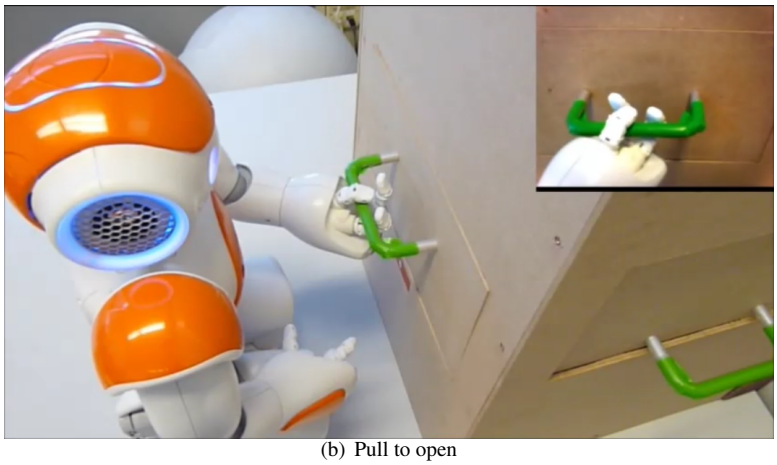
The state of a garbage can was described by the size of its openness. To detect the opened area, we put a black plastic bag in each garbage can and calculated the area of the dark part in an captured image. The opened area was also located by a bounding box with a size of $w \times h$ in pixels. Then, s_o was the percentage of opened area in an image:

$$s_o = \frac{w \times h}{W \times H} \tag{4.19}$$

where $0 \leq s_o \leq 1$.



(a) Push to open



(b) Pull to open

Figure 4.3: An illustration of different actions (push and pull) and their effects on the lid opening. Note that these motor skills are to be learned by the robot rather than preprogrammed. A video is available at http://youtu.be/oluLDwMaVoY. The sub-images at the up right corner of (a) and (b) show the view from the NAO's camera.

At each time step, a robot action a was selected from $\mathcal{A} = \{(x, y, z)^T \in \mathbb{R}^3 | -0.01 \leq x, y, z \leq 0.01\}$ (in meters)[4]. After an action was performed, the robot captured another image and obtained the new state of object s'_o in the same way as (5.14). The effect was obtained as follows:

$$e_o = s'_o - s_o \tag{4.20}$$

To approximate each forward model in equation (4.1), we used a feed-forward neural network with four input neurons (one neuron for s_o and three neurons for a), one hidden layer with 10 neurons and one output neuron for e_o. We also used two neural networks to approximate the actor and critic. We normalized the action values to $[-1, 1]$ in each dimension. For the three layers of all neural networks, we used linear, hyperbolic tangent and linear transfer functions, respectively. All weights of neural networks were initialized randomly in $[-0.3, 0.3]$. The learning rates in equation (4.4), (4.8) and (4.9) were $\alpha = 0.3$, $\zeta = 0.3$, $\beta = 0.3$. The Gaussian exploration parameter in equation (4.5) was $\delta = 0.2$ for each action dimension. The discount factor in equation (4.7) was $\gamma = 0.9$. The TD errors were averaged over $N = 20$ actions in equation (4.10) and $\epsilon = 1 \times 10^{-4}$.

We tested the active exploration approach against the baseline of random exploration. In the case of random exploration, we used a random actor and its output was a random number in $\mathcal{A}$. We ran experiments in both 1D action space (X axis of NAO space) and 3D Cartesian space for the two garbage cans. In all experiments, the garbage cans were initially closed. We set $N = 20$ in equation (4.11), and NAO performed the first 20 actions randomly. Then, it continued random exploration or switched to the active learning mode. The maximal allowed exploration steps were $N_s = 100$ and $N_s = 300$ for the 1D action space and 3D action space, respectively.

4.6.2 Results

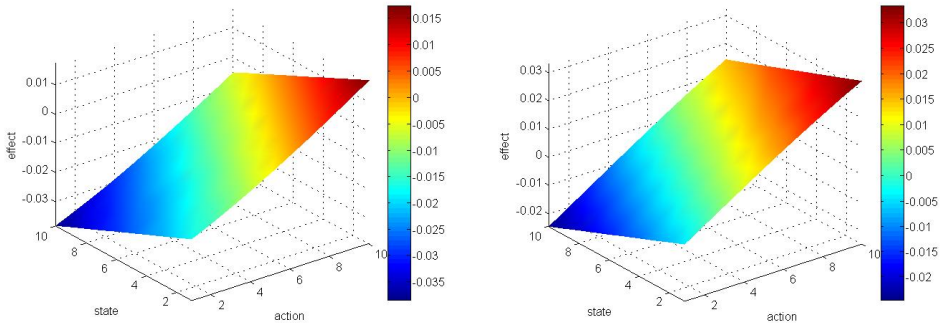
Learned Forward Models

For purpose of visualization, we have plotted the forward models learned in 1D state space and 1D action space (see Figure 4.4 and Figure 4.5).

For both the push-lid and the pull-handle, we bound the state space to $[0, 0.2]$ which covers object states from closed-lid to open-lid with the size of openness 0.2 (see equation (5.14) for the definition of openness). For any given object state, the action space $\mathcal{A}$ is $[-0.01, 0.01]$ (in meters). They are meshed into 10×10 grids for plotting the surface of predicted effects. For example, state = 0 corresponds to zero openness, i.e., a closed lid, state = 10 corresponds to openness 0.2; similarly, action = 0 corresponds to $a = (-0.01, 0, 0)^T$ (contract arm backward), and action = 10 corresponds to $a = (0.01, 0, 0)^T$ (stretch arm forward). In other words, the robot can choose any actions from the continuous space $[-0.01, 0.01]$ in any given object state within the continuous space $[0, 0.2]$. The effect also takes value in continuous space. A positive value corresponds to lid opening effect, a negative value corresponds to lid closing effect, and effect = 0 means no effect.

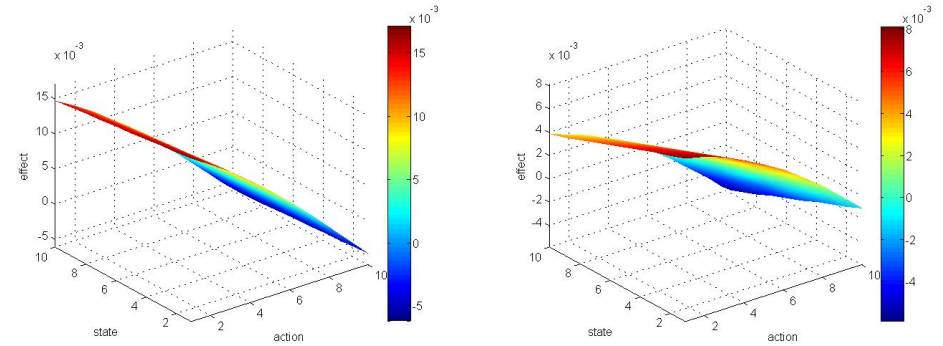
Both Figure 4.4 and Figure 4.5 show the near linear relations between object states, robot actions and the predicted effects. For the push-lid (see Figure 4.4), stretching the

[4] In the Cartesian space of NAO, the X axis is positive toward NAO's front, the Y from right to left and the Z is vertical (see Figure 3.2). At each time step, an action was constrained in $\mathcal{A}$ to avoid potential motor damage, e.g., pushing too hard against an unmovable object.



(a) random exploration (b) active exploration

Figure 4.4: The learned forward models with the push-lid in 1D state space and 1D action space using (a) random exploration, (b) active exploration ($0 \leq$ state ≤ 0.2, $-0.01 \leq$ action ≤ 0.01).



(a) random exploration (b) active exploration

Figure 4.5: The learned forward models with the pull-handle in 1D state space and 1D action space using (a) random exploration, (b) active exploration ($0 \leq$ state ≤ 0.2, $-0.01 \leq$ action ≤ 0.01).

arm (action > 5) is likely to result in the opening effect (effect > 0), and stretching further would result in more opening. Besides, the opening effect decreases when the current state of opening increases. This prediction agrees with the hinged design of the push-lid. Similarly, the closing effect is predicted when contracting the arm (action < 5). Both forward models, either learned through random exploration (Figure 4.4(a)) or active exploration (Figure 4.4(a)), are able to predict the opening and closing effect. For the pull-handle (see Figure 4.5), in contrast, stretching the arm (action > 5) is likely to result in the closing effect (effect < 0), and contracting the arm would result in the opening effect (effect > 0). This also agrees with the hinged design of the lid.

In both groups of "random exploration versus active exploration", all forward models can approximately predict the trend of lid opening or closing. However, the predicted effect values are different for a same state and a same action between the two models within the same group. This difference is caused by the distribution of collected data in the continuous state and action spaces. In the case of random exploration, random actions are distributed evenly in $[-0.01, 0.01]$, which means that the robot stretches or contracts its arm with the same chance. As the initial state of a garbage can is always closed, many data points are likely to be collected near the initial object state. Moreover, some actions are very close to 0, which result in insignificant movement of the arm and insignificant state change of the garbage can. Consequently, such data contributes little to the learning of the forward model. In contrast, active exploration prefers to select actions that have resulted in high effect prediction errors. These actions are enforced to be selected and the Gaussian exploration is centered around these actions. In this way, the actions have a high chance to be close to the boundaries of the action space, i.e., -0.01 or 0.01. In the experiments, the robot continued to stretch its arm until the arm was fully stretched, and then it started to contract the arm. This process went on until the termination condition was satisfied. As a result, the state space was explored more thoroughly than random exploration within the same number of action steps.

Comparison of Averaged Absolute TD errors

The averaged absolute TD errors (see equation (4.11)) are compared in Figure 4.6 - 4.9 between random exploration and active exploration in 1D and 3D action spaces. In all experiments, they converge for active exploration while they fail to converge for random exploration within allowed number of action steps.

In the active exploration mode, the robot intends to explore the most uncertain spaces in an organized way. It is likely to end up with its arm fully stretched or contracted, i.e., when a lid is maximally opened or tightly closed. In this case, the object state becomes stable and no more effect is observed, which gives the TD errors good chance to converge. In contrast, the random exploration is less efficient. It occasionally runs into situations with high prediction errors so that the TD errors would hardly converge within given action steps. Active exploration requires much less data samples under the proposed criteria.

We note that the TD errors converge much faster for active exploration in 1D action space than in 3D action space for both garbage cans. Intuitively, the dimension of an action space influences the convergence of TD errors. The higher the dimension, the more difficult the learning problem is. Consequently, it would take more exploratory actions to learn an accurate forward model.

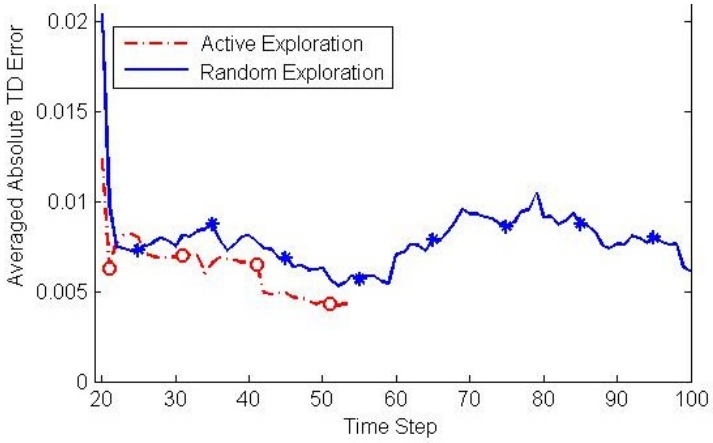


Figure 4.6: Experimental result with the push-lid in 1D action space.

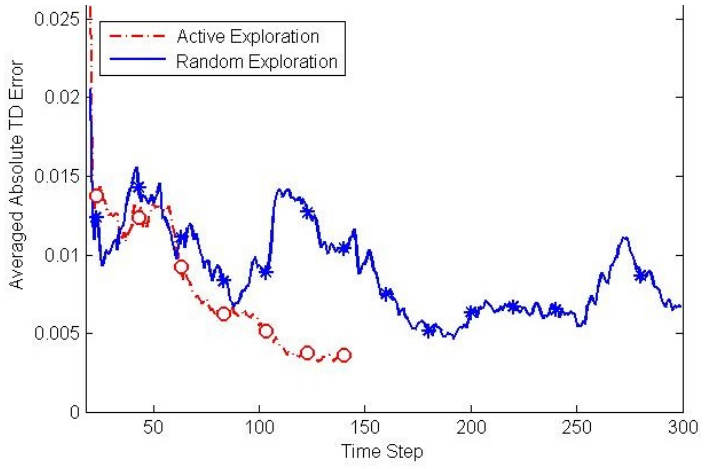


Figure 4.7: Experimental result with the push-lid in 3D action space.

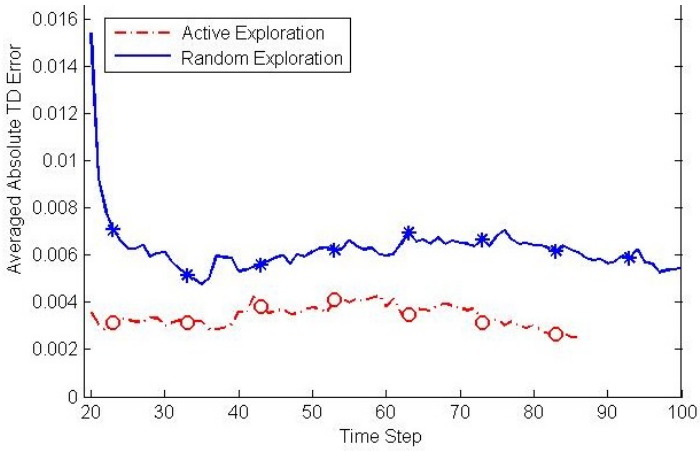


Figure 4.8: Experimental result with the pull-handle in 1D action space.

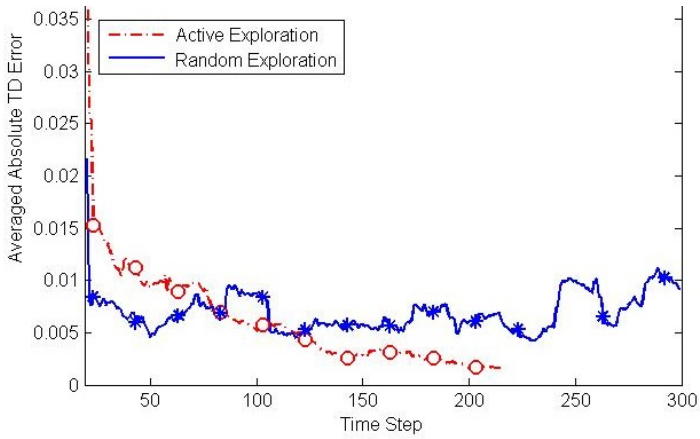


Figure 4.9: Experimental result with the pull-handle in 3D action space.

Skill Acquisition

The acquired motor skills using 8 forward models (FM) are shown in Table 4.1.

The 8 forward models correspond to the 8 combinations of two garbage cans (push-lid ψ_l and pull-handle ψ_h), two action exploration modes (random and active), and two action spaces (1D and 3D). In all cases, skill learning started with a tightly closed lid ($s_o = 0$). A sequence of actions were selected to maximize the opening effect first, then minimizing the opening effect. We set $\tau = 0.01$ in equation (4.15). Consequently, 16 action sequences were performed to open and close the garbage cans. We list the first and the last actions in each action sequence. Besides, all values are rounded to three decimal points.

In the first two cases of the push-lid (FM 1 and FM 2), the same sequence of 6 pushing actions $a = (0.008, 0, 0)$ were performed until the lid was maximally opened ($|e_o| < \tau$), with the robot arm fully stretched. In this way, a "push lid open" skill has been acquired. Then, the robot started the acquisition of a "close lid" skill by contracting its arm, i.e., $a = (-0.01, 0, 0)$, until no more state change was observed ($|e_o| < \tau$). We note that the push-lid could not be closed completely by NAO due to the friction between the lid and the body part. This happened in all 4 experiments with the push-lid.

In the other two cases of the push-lid (FM 3 and FM 4), the action sequences for lid opening are different with the 1D cases. NAO not only stretched its left arm forward, but also moved the arm left and upward, e.g., $a = (0.008, 0.010, 0.010)$ proposed by FM 3, and $a = (0.008, 0.008, 0.002)$ proposed by FM 4. Because such movements would increase the camera view of observed size of openness.

In all the four cases of pull-handle (FM 5, FM 6, FM 7, FM 8), NAO have acquired the skills to pull the handle for lid opening and push the handle for lid closing. Also, the 3D actions are different from the 1D actions. When pulling the handle, the arm also moved downward, e.g., $a = (-0.010, 0, -0.010)$ proposed by FM 7, and $a = (-0.010, 0, -0.008)$ proposed by FM 8. This result agrees with the constrained movement of the handle.

4.7 Conclusions and Open Issues

This chapter proposed an architecture for active affordance learning in continuous state and action spaces along with manipulation skill acquisition. We have used function approximation to model and generalize affordances in the continuous spaces. Affordance models can be learned using the prediction error of action effects. The prediction error also serves as a reward signal to update the action exploration policy using an actor-critic reinforcement learning structure. The termination of affordance learning is based on the convergence of the actor-critic module. Then, the learned affordance models can be reused to acquire a range of manipulation skills by generating and achieving goals.

We have carried out an experimental evaluation of the proposed active learning architecture. Feed-forward neural networks have been used to model affordances, and continuous actor-critic learning automation (CACLA) has been chosen to learn the action exploration policy. The proposed active affordance learning algorithm has shown a good performance. Compared with the commonly used random action exploration baseline, our algorithm has required less exploratory actions for the convergence of the averaged absolute TD errors in both the one dimensional and three dimensional action space cases. This indicates that

Table 4.1: The acquired motor skills in different states (s_o) using 8 forward models (FM), 1D or 3D actions (a), self-generated goals (e_o^g), and observed effects (e_o).

FM	ψ	policy	s_o	a (m)	e_o^g	e_o
1	ψ_l	random	0	(0.008, 0, 0)	0.018	0.009
			0.114	(0.008, 0, 0)	0.003	0.002
		1D	0.117	(-0.010, 0, 0)	-0.031	-0.009
			0.018	(-0.010, 0, 0)	-0.020	0
2	ψ_l	active	0	(0.008, 0, 0)	0.033	0.025
			0.124	(0.008, 0, 0)	0.020	0.004
		1D	0.135	(-0.010, 0, 0)	-0.019	-0.002
			0.035	(-0.010, 0, 0)	-0.008	0
3	ψ_l	random	0	(0.008,0.008,0.002)	0.034	0.016
			0.072	(0.008, 0.008, 0)	0.024	0
		3D	0.071	(-0.010, 0.008, -0.010)	-0.012	-0.021
			0.005	(-0.010, 0.008, -0.010)	-0.005	-0.001
4	ψ_l	active	0	(0.008, 0.010, 0.010)	0.031	0.019
			0.112	(0.008, 0.010, 0.010)	0.021	0
		3D	0.094	(-0.010, -0.010, -0.010)	-0.044	-0.035
			0.024	(-0.010, -0.010, -0.010)	-0.039	0
5	ψ_h	random	0	(-0.010, 0, 0)	0.008	0.007
			0.134	(0.008, 0, 0)	0.022	0.016
		1D	0.135	(-0.010, 0, 0)	-0.019	-0.017
			0.035	(-0.010, 0, 0)	-0.008	0
6	ψ_h	active	0	(-0.010, 0, 0)	0.017	0.011
			0.101	(-0.010, 0, 0)	0.014	0
		1D	0.101	(0.008, 0, 0)	-0.006	-0.044
			0	(0.008, 0, 0)	-0.004	0.001
7	ψ_h	random	0	(-0.010, 0, -0.010)	0.017	0.007
			0.119	(-0.010, 0, -0.010)	0.010	0.001
		3D	0.123	(0.008, 0, 0.008)	-0.019	-0.038
			0	(0.008, 0, 0.008)	-0.012	0
8	ψ_h	active	0.012	(-0.010, 0, -0.008)	0.009	0.012
			0.094	(-0.010, 0, -0.008)	0.012	0.001
		3D	0.095	(0.008, 0, 0.008)	-0.003	0.001
			0	(0.008, 0, 0.008)	-0.003	0

the active exploration is more efficient than random exploration. The reason is that active exploration is carried out in a more organized way by sampling in the most uncertain areas. In contrast, the random exploration occasionally samples an action resulting in high TD error which makes the averaged absolute TD errors more difficult to converge. Besides, the experimental results have shown that the number of exploratory actions needed increases with the dimension of the action space. In other words, the higher the dimension of the action space, the more difficult the learning problem is. It will be important to consider the scalability of the active affordance learning algorithm to further speed up the learning in higher dimensional action spaces.

In the model exploitation component, manipulation skills are acquired automatically by generating a sequence of goals in the effect space and selecting actions to achieve the goals. Using the learned affordance models, the goal generation process continues until there is no significant effect observed. Meanwhile, corresponding actions are selected to achieve the goals. In the garbage can manipulation task, a humanoid robot NAO has successfully learned how to open and close two types of garbage cans. These skills are associated with initial states and goal states, therefore they are immediately reusable when other goal-directed tasks are given later on. Unlike Chapter 3 addressed with on-line model learning and use, this chapter did not consider model learning in the model exploitation component. Nevertheless, it is still possible to enable the interaction between the model learning component and the model exploitation component. It is an interesting research topic to combine active generation of goals [119] and active exploration in the action space for affordance learning.

The proposed architecture is generic for active affordance learning and is flexible for implementation. Trying and comparing different instantiations of the modules in the architecture is an immediate extension of the current research. The architecture provides support for any perceptual proxy for representing objects, action space for representing actions, and forward model for representing affordances in continuous spaces. We have chosen the 2D bounding box perceptual proxy, the 3D Cartesian action space and the feed-forward neural networks, respectively. These are not necessarily the best choices. It is possible to use more sophisticated representations of objects such as 3D point clouds [117], actions in high-dimensional joint space [119], and forward models that require less data to train. In addition, any actor-critic RL algorithm can be chosen for the active exploration component as long as the algorithm supports continuous states and actions. It would be interesting to compare the results of various actor-critic RL algorithms. Moreover, other intrinsically motivated RL algorithms can be employed if they also make use of prediction errors as the reward signals, such as intelligent adaptive curiosity (IAC) [113] and robust intelligent adaptive curiosity (R-IAC) [112].

Finally, this chapter has considered affordance learning of a single object. We have chosen to use a perceptual proxy representation of objects [38] which provides the opportunity to generalize the relations between object representation and object manipulation. This enables to transfer learned affordances across objects. As a result, the acquired manipulation skills can be transferred to handle new objects. We will further discuss this in Chapter 5.

CHAPTER FIVE

Transfer Learning of Affordances for Complex Objects

This chapter introduces a robot learning architecture that enables the transfer of learned affordances to speed up the learning of a new object. We develop a part-based affordance model for handling household objects that are composed of several parts. Through embodied robot interaction with every object part, the robot can learn affordance models to predict the action effects of all the parts. As a result, functional part(s) can be chosen to achieve desired effects when a goal is given. This chapter addresses the challenge of speeding up the identification of functional part(s) of a new object by referring to relevant known objects. The selection of relevant objects is based on the measure of object similarities, and the prediction of a functional part is based on the use of the corresponding affordance models. Potential functional part(s) of the new object can be found along with the associated manipulation skills. Besides, the robot not only evaluates by itself the actual outcomes of the prediction of affordance models, but also adjusts by itself its action selection strategy if necessary. We demonstrate through real-world experiments with the humanoid robot NAO that our method is able to speed up the learning of a new type of garbage can by transferring the affordances learned previously from similar garbage cans.

5.1 Introduction

In the previous chapter, we have proposed an active affordance learning approach in continuous state and action spaces. It deals with only a single object at a time. However, learning every object from scratch is not efficient because it takes time for a robot to perform exploratory actions on objects. The current chapter considers multiple objects and their similarities. The main motivation is to speed up the learning of how to manipulate a new object by transferring the robot's past interaction experience with relevant objects. In other words, we aim at reducing the number of exploratory actions for learning how to manipulate the new object. For instance, the robot may speed up the learning of how to open an oven by transferring the learned affordances with doors, refrigerators, drawers, cabinets, etc.

In the literature, *transfer learning* [120] is defined as the improvement of learning in a

target task through the transfer of knowledge from related *source task(s)*. In this chapter, we combine the concepts of transfer learning and affordance to speed up the learning of a target task. For the ease of description, the term *source object(s)* refers to the object(s) that have been encountered in the course of learning the source task(s), and *target object* refers to the object that has been encountered in the target task. We define transfer learning of affordances as robot learning of a target task with data and additional sources of information: affordances related to one or more source objects (see Figure 5.1).

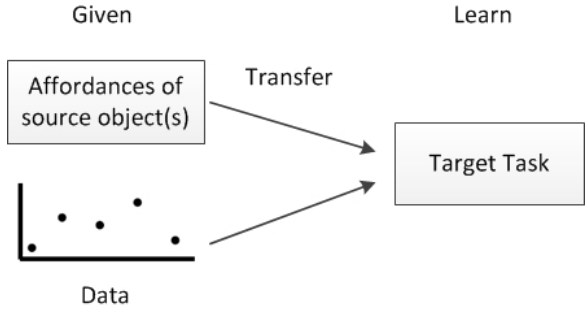


Figure 5.1: An illustration of transfer learning of affordances.

Reusing learned affordances has been considered to solve a target task. Typically, the target object is first classified into one category of the source objects, and the action effects are predicted according to the learned affordances of the source objects. Then, action selection is carried out to achieve desired effects with the target object. For example, a tapping action is selected to make a ball roll in an imitation task [10]. In [33], a sequence of actions are selected to push and position a target object to a goal location. However, only simple toy objects have been considered in the literature, and a fixed set of discrete actions have been used. This chapter considers complex household objects composed of several parts, along with parameterized robot actions in continuous spaces. In addition, knowledge transfer in the literature simply assumes that the previously effective actions are still effective for similar objects. There is no learning in the target task. In contrast, this chapter proposes transfer learning of affordances to structure the search for effective actions for the target object.

The main contribution of this chapter is that we propose a robot learning architecture that improves the learning of a target object through knowledge transfer from similar source objects. We extend affordance learning of individual objects to a multi-object scenario. A part-based affordance model is proposed, and object similarity is measured by comparing the features of object parts. For each object part, a regression model is learned to approximate the relation between actions and effects in continuous spaces. By reusing the corresponding affordances of the source objects, the robot actively selects object parts to interact with, and evaluates the actual action effects on-line. In case that the knowledge transfer does not work as anticipated, the robot changes its exploration strategy accordingly. This prevents the robot from being stuck with a wrong action, e.g., keep pulling a door handle which has to be turned.

The remainder of this chapter is structured as follows. Section 5.2 gives a brief overview

of the literature related to our approach. Section 5.3 describes the part-based affordance model, followed by the affordance learning and use approaches in Section 5.4. Section 5.5 proposes the transfer learning method. Section 5.6 introduces the task environment for evaluation and the experimental results. Section 5.7 concludes the chapter.

5.2 Related Work

In machine learning, transfer learning approaches have close relation with task learning algorithms [120]. One approach of transfer learning is in the context of inductive learning, where classification and inference algorithms are extended. Typically, a predictive model is learned with generalization capability from a set of training data, whose true distribution is assumed with an inductive bias by the learning algorithm. Transfer learning works by affecting the inductive bias of the target task with the knowledge from source tasks. It aims to speed up the model learning or to improve its generalization capability. For example, the search for neural network parameters of a target task is sped up by using the slope information obtained with gradient-descent algorithm in the source tasks [121]. For Bayesian learning methods, source-task knowledge is naturally incorporated to produce an informative prior distribution that combines the target-task data to make predictions [122, 123]. As we have modeled affordance learning as learning a predictive affordance model, transfer learning of affordances is related to the inductive transfer approach. Specifically, we reuse affordance models learned in the source tasks to select actions to achieve the desired effect in the target task.

Reinforcement Learning (RL) is another major context of transfer learning. Transfer methods can be distinguished according to several aspects such as assumptions about source task selection, allowed task differences, identification of transferable information, metrics to measure successful transfer, etc. [124]. Transfer in RL aims to speed up the learning process, since RL may take many exploratory episodes before acquiring a reasonable value function or policy. One straightforward method of transfer in RL is to set the initial solution in a target task based on knowledge from a source task. In a starting-point method [125] used for transfer in temporal-difference RL, the final value function of the source task is used as the initial solution for the target task. Another type of RL transfer involves imitation of the source task policy for action selection while learning the target task. The intuition is that the learned policy in a source task is more informative than the random exploration policy or ill-formed exploitation in the early steps of the target task [126]. These methods can be used to transfer the value functions or policies discussed in chapter 3 and 4. However, this chapter focuses on the inductive transfer approach, and we leave the RL transfer approach for future work. Nevertheless, we take inspirations from the RL transfer approach. Specifically, we will discuss the selection of source tasks and the measure of a successful transfer.

5.3 Part-based Affordance Model

We follow the definition of affordance model in section 3.2. In this chapter, we extend the affordance definition (3.1) to the case of objects composed of several parts. Throughout

this chapter, we discuss transfer learning of affordances by using the example of lid opening task (see Fig. 5.2).

5.3.1 Perception of Objects, Parts and States

The robot perceives its environment and extracts visual features from its camera image. We assume that the robot can identify object parts based on known features (markers in our experiments) and color segmentation to retrieve connected components of 2D pixels called blobs [127]. Each blob corresponds to an object part and is bounded by a minimum-enclosing rectangle (see Fig. 5.2). Based on a segmented blob and its bounding box, we use shape descriptors to describe each part of an object (see Table 5.1).

Table 5.1: Shape descriptors.

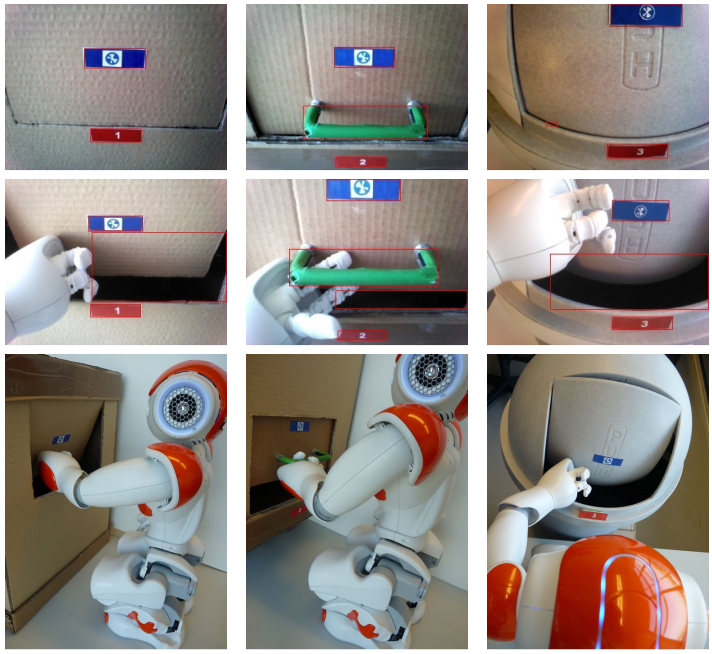
Descriptor	Definition
area	ratio between pixel number of the blob and pixel number of the image
rectangleness	ratio between the short side and long side of the bounding box
squareness	ratio between blob area and bounding box area

The robot then recognizes a household object as a combination of its parts, without necessarily knowing which of them are essential for the given task. Refer to [127] for more 2D shape descriptors, and refer to [117] for a more advanced method to recognize object parts with a RGB-D camera.

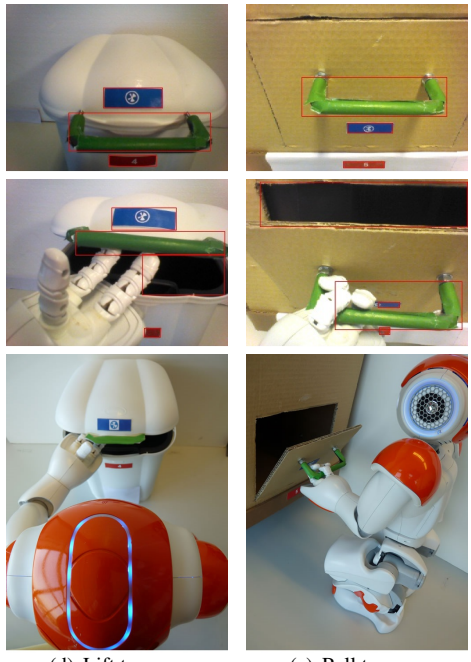
Denote by O the set of objects, by Ψ the set of all known object parts (container, lid, handle, pedal, etc.). As not all objects necessarily contain the same parts, we denote by $\Psi_o \subseteq \Psi$ the set of parts that object $o \in O$ is composed of. For an object o and its part $\psi \in \Psi_o$, we use $s_o \in S_o$ and $s_\psi \in S_\psi$ to denote the state of the object and the part, respectively. The states change with time and are continuously measured by robot's sensors. In our case, s_o is the current size of the garbage can opening, and s_ψ is the current position of the lid or handle. The time index is omitted for the ease of notation.

5.3.2 Robot Actions

Denote by $a(\theta)$ an action parameterized by a real parameter vector θ, where $a \in \mathcal{A}$ is the type of action (push, pull, lift, etc.) and $\theta = (x, y, z)^T \in \Theta \subseteq \mathbb{R}^3$ is the action parameter that takes real values in the 3D Cartesian space. The robot always approaches the vicinity of an object's part before interacting with it. We note that the the parametrized actions defined in this section is not the same with the actions defined in section 4.4.1. The actions defined in the previous chapter are in random direction. In contrast, the action type a indicates the general direction of the action, e.g., push is moving the arm forwards, and pull is moving the arm backwards. The action type can be defined according to the skills acquired in section 4.5. The action parameter θ is also important which indicates how much an action should be performed. As household objects come with different designs, it will be useful to learn the action parameters to handle the objects properly. For example, the garbage cans with different push-lids are opened wide enough with different push parameters (see Figure 5.2(a) and Figure 5.2(c)).



(a) Push to open (b) Lift to open (c) Push to open



(d) Lift to open (e) Pull to open

Figure 5.2: An illustration of different action types (push, lift and pull) and their effects in the lid opening task.

5.3.3 Action Effects

The effects of action $a(\theta)$ on part $\psi \in \Psi_o$ and object o are denoted by $e_\psi \in \mathcal{E}_\psi$ and $e_o \in \mathcal{E}_o$, respectively. They are measured by

$$e_\psi = m_1(s_\psi, s'_\psi) \tag{5.1}$$

and

$$e_o = m_2(s_o, s'_o) \tag{5.2}$$

where s'_ψ and s'_o are the states of ψ and o after $a(\theta)$ was applied, m_1 and m_2 are suitable metrics. For example, lifting a handle results in the displacement of the handle as well as the change of the opening size (see Fig. 5.2(b)). In this chapter, the effect spaces are $\mathcal{E}_\psi \subseteq \mathbb{R}$ and $\mathcal{E}_o \subseteq \mathbb{R}$ and m_2 simply is subtracting the previous state from the new state:

$$e_o = s'_o - s_o \tag{5.3}$$

5.4 A Baseline Without Transfer Learning

We assume that the robot already has a plan for executing a task with a given household object. The task plan consists of an initial condition and a termination condition. For example, in the case of lid opening, a lid is initially closed and has to be opened wide enough. The same as the previous chapters, we define affordance learning as the process of learning prediction models of action effects except that we take into account object parts in this chapter.

5.4.1 Functional Parts

We call a part *functional* if it can be manipulated to achieve a desired effect. For example, the lids in Figure 5.2(a) and Figure 5.2(c) are functional because they can be pushed open. The handles in Figure 5.2(b), Figure 5.2(d) and Figure 5.2(e) are functional because they can be lifted/pulled to open the lids. The bodies of the garbage cans are also functional if they are pushable to move the garbage cans to desired locations.

During affordance learning with an object, the robot interacts with each part of the object without knowing which part is functional. Later on, the learned affordances can be used to select a part as well as to select a corresponding action to achieve a desired effect (see Table 5.2).

Table 5.2: Affordance learning and use based on relations between object (P)arts, (A)ctions, and (E)ffects.

Inputs	Outputs	Functions
(P, A)	E	Learn prediction models
E	(P, A)	Select a functional part and an action

5.4.2 Learn Object-Part Relation

Prediction Models

In order to learn affordances of each part ψ of an object o, the robot needs to learn both the effect of performing an action on a part and also what this effect does to the object's state. First, a prediction model g_o maps from the parameterized action space $\mathcal{A} \times \Theta$ to the effect space $\mathcal{E}_\psi$ for each part $\psi \in \Psi_o$:

$$g_o : \mathcal{A} \times \Theta \rightarrow \mathcal{E}_\psi \tag{5.4}$$

The model g_o captures the relationship between a parameterized action and the effect on ψ. Besides, a mapping h_a models the relation between the effect spaces $\mathcal{E}_\psi$ and $\mathcal{E}_o$:

$$h_a : \mathcal{E}_\psi \rightarrow \mathcal{E}_o \tag{5.5}$$

In other words, the model h_a captures the functional relationship between a part ψ and an object o.

Data collection

Whenever an effective interaction between the robot and an object part ψ took place, the following data tuple is collected:

$$(o, s_o, \psi, s_\psi, a, \theta, e_\psi, e_o) \tag{5.6}$$

where $o \in \mathcal{O}$, $\psi \in \Psi_o$ and $a \in \mathcal{A}$ take discrete values, while $s_o \in S_o$, $s_\psi \in S_\psi$, $\theta \in \Theta$, $e_\psi \in \mathcal{E}_\psi$ and $e_o \in \mathcal{E}_o$ take real values in continuous spaces.

In an exploration stage, an action of type a is typically carried out maximally n times on a given part $\psi \in \Psi_o$ with various parameters selected from Θ. The learned model (5.4) is allowed to be generalized in Θ for effect prediction. Besides, data collection for a pair (ψ, a) will stop if the learned model is accurate enough to predict the effect of a on ψ. Denote by $\mathcal{D}_{o,\psi,a}(k), k \leq n$ the set of collected data (5.6) after having performed $a(\theta_1), ..., a(\theta_k)$ on ψ.

Function approximation

By using $\mathcal{D}_{o,\psi,a}(n)$, the prediction models (5.4) and (5.5) can be learned as:

$$e_\psi = \hat{g}_o(a, \theta) \tag{5.7}$$

and

$$e_o = \hat{h}_a(e_\psi) \tag{5.8}$$

where $\hat{g}_o$ and $\hat{h}_a$ are the approximation of g_o and h_a. Standard regression methods, e.g., linear regression [64] or Gaussian Process Regression [128], can be used to obtain $\hat{g}_o$ and $\hat{h}_a$.

5.4.3 Part Selection and Action Selection

Denote by $s_o^g \in S_o$ the goal object state. We assume that s_o^g is known and a specific action that can achieve s_o^g exists. In a given task, s_o^g can be hard coded or provided to the robot by demonstration [33]. The task is terminated by the following condition:

$$\|s_o - s_o^g\| \leq \delta_1 \tag{5.9}$$

where $s_o \in S_o$ is the current object state, $\|.\|$ is the distance metric in state space S_o, and δ_1 is a positive real value.

Given an initial object state s_o^i, the robot aims to select a part ψ^* and an action $a^*(\theta^*)$ to achieve the goal state s_o^g, i.e., the condition (5.9). Combining (5.3), (5.8) and (5.9), the following condition should be satisfied:

$$\|\hat{h}_a(e_\psi) - (s_o^g - s_o^i)\| \leq \delta_1 \tag{5.10}$$

First, the robot selects a pair of part and action type (ψ^*, a^*) along with the associated e_ψ which are most likely to result in (5.10):

$$(\psi^*, a^*, e_\psi^*) = \underset{\psi \in \Psi_o, a \in \mathcal{A}, e_\psi \in \mathcal{E}_\psi}{\arg\min} \|\hat{h}_a(e_\psi) - (s_o^g - s_o^i)\| \tag{5.11}$$

Then, the action parameter θ^* is selected which corresponds to the minimal change of the end-effector in the action parameter space:

$$\theta^* = \underset{\theta \in \Theta}{\arg\min}\{\|\theta\| \;\big|\; \|\hat{g}_o(a^*, \theta) - e_\psi^*\| \leq \delta_2\} \tag{5.12}$$

where δ_2 is a positive real value.

The above process of affordance learning and use is summarized in Algorithm 4.

Algorithm 4: Affordance learning and use with an object.

inputs A set of object parts Ψ_o, an initial object state s_o^i and a goal object state s_o^g;
outputs $\psi^* \in \Psi_o$, $a^* \in \mathcal{A}$, $\theta^* \in \Theta$;

```
for all ψ ∈ Ψ_o do
    for all a ∈ A do
        k = 1;
        while k ≤ n do
            Select θ_k ∈ Θ_exp randomly, apply a(θ_k) on part ψ, observe effects and
            collect data sample (5.6) ;
            Learn (5.7) and (5.8) based on data D_{o,ψ,a}(k);
            k ← k + 1
        end while
    end for
end for
Select ψ* and a* using (5.11);
Select θ* using (5.12);
```

Finally, $a^*(\theta^*)$ is performed on ψ^*. If (5.9) is satisfied, the learning is considered a success; otherwise, the learning continues by collecting more data samples (5.6) or terminates after a certain number of actions.

5.5 Transfer Learning of Affordances

Denote by o_{tar} the target object, and O_{kn} the set of source objects.

5.5.1 Why to Transfer

Affordance learning in Algorithm 4 involves robot interaction with every part $\psi \in \Psi_{o_{tar}}$ using every action type $a \in \mathcal{A}$. The required number of exploratory actions is calculated as follows:

$$N(o_{tar}) = \sum_{\psi \in \Psi_{o_{tar}}, a \in \mathcal{A}} \left| \mathcal{D}_{o_{tar},\psi,a}(n) \right| \tag{5.13}$$

where $|.|$ is the Cardinality of a set, and $\mathcal{D}_{o_{tar},\psi,a}(n)$ is the data set of applying action a on part ψ. The more parts the target object o_{tar} has, and the more action types the robot has, the more exploratory actions are needed. For a given part ψ and an action type a, sufficient data have to be sampled in the action parameter space Θ. The higher the dimension of Θ, the bigger the data set $\mathcal{D}_{o_{tar},\psi,a}(n)$. Therefore, it can take a very long time to learn all the prediction models (5.7) and (5.8) before the goal condition (5.9) is achieved.

We propose that $N(o_{tar})$ can be reduced through transfer learning of affordances. The purpose is not to learn every prediction model $\hat{g}_o$ or $\hat{h}_a$, but is to achieve the goal condition (5.9) as fast as possible. By taking reference of how O_{kn} were manipulated in the past, the robot does not have to try every part $\psi \in \Psi_{o_{tar}}$ or every action type $a \in \mathcal{A}$. It may be beneficial to initialize the learning by selecting a part from $\Psi_{o_{tar}}$ which looks similar to a functional part of O_{kn}. The corresponding action type and action parameter can also be actively selected rather than randomly selected. If lucky, the functional part of o_{tar} along with the parameterized action will be found faster than exploring randomly in $\Psi_{o_{tar}}$, $\mathcal{A}$ and Θ. Even if the learned affordances with O_{kn} are not helpful for the target task, $N(o_{tar})$ will be no more than the case without any knowledge transfer.

5.5.2 Transfer Learning Architecture

In the sequel, we develop a framework for learning about the target object through transfer learning of affordances. The main challenge is the selection of source objects that are relevant to the target object, as well as how the affordances of the source objects can contribute to the target task. The overall architecture for transfer learning of affordances is illustrated in Figure 5.3.

The Action Selection module is at the core of the transfer learning architecture. The robot is provided with the initial and termination conditions for the target task, e.g., the initial and goal states for the lid opening task. Action selection continues until the task goal is achieved. Without transfer learning, the robot has to select exploratory actions to learn affordances (exploration) before it can use the learned affordances to select goal-directed actions (exploitation). Through transfer learning, the robot can choose either the exploration mode or the exploitation mode by reusing the affordances of the source objects. We propose that the robot starts with the exploitation mode and continues to explore/exploit alternatively until the goal is achieved. It is an aggressive strategy that the robot always tries to exploit the learned affordances to select goal-directed actions whenever possible.

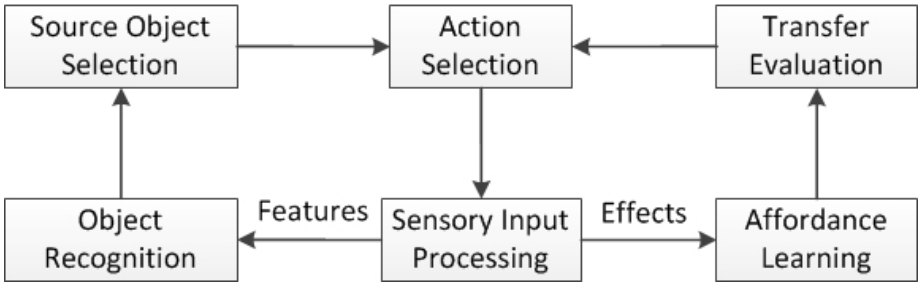


Figure 5.3: An architecture for transfer learning of affordances.

Before taking any actions, the robot first observes the target object o_{tar} and processes the sensory input in the Sensory Input Processing module. The Object Recognition module is responsible for recognizing the set of object parts $\Psi_{o_{tar}}$. Then, the database of known objects is queried to search for source objects that are likely to be relevant to the target task. A set of source objects O_{src} is selected in the Source Object Selection module. The selection of O_{src} is based on measuring the similarity of object parts. Those objects which have functional parts that look similar to a part of o_{tar} are added to O_{src}. We will further discuss the selection of source objects in section 5.5.3.

If O_{src} is not an empty set, it means that there exists at least one source object. Initially, the robot selects the exploitation mode and transfers the past experience to select goal-directed actions. The intuition is straightforward: the effective action for the functional part of a source object is anticipated to be still effective for the similar part of the target object. This action along with the action parameter can be found by reusing the learned affordance models of the source object. After performing the action, the robot observes the actual effect. If the anticipated effect is not achieved, the robot can continue the exploitation mode by referring to other source objects. Otherwise, the robot switches to the exploration mode, i.e., the Affordance Learning module learns the predictive affordance models as detailed in section 5.4.2. After having collected some data about o_{tar}, the Transfer Evaluation module evaluates whether the transfer of affordances is successful or not. This prevents the robot from following a wrong action selection policy. We will further discuss how the affordances of the source objects are transferred in section 5.5.4.

5.5.3 Source Object Selection

The robot has to decide by itself whether the transfer of affordances should happen. This decision is made in the Source Object Selection module, whose output is a set of source objects O_{src}. If O_{src} was an empty set, there would be no source objects for transfer learning; otherwise, the transfer learning would be initiated. Algorithm 5 describes how O_{src} is decided.

Lines 2 through 11 search for the set of source objects O_{src} from a set of known objects O_{kn}. For each known object $o \in O_{kn}$, it takes two steps to decide whether o is relevant to the target task.

The first step is to find the functional part ψ^* of o. A task rehearsal is carried out for

Algorithm 5: Find relevant source objects.

inputs A set of known objects O_{kn} and a target object o_{tar};
An initial object state $s^i_{o_{tar}}$ and a goal state $s^g_{o_{tar}}$;
Learned models $e_o = \hat{h}_a(e_\psi)$, where $o \in O_{kn}$, $\psi \in \Psi_o$, $a \in \mathcal{A}$;
outputs A subset of objects $O_{src} \subset O_{kn}$ sorted by similarity with o_{tar};

1: Initialize $O_{src} = \emptyset$;
2: Observe o_{tar} and obtain the set of its parts $\Psi_{o_{tar}}$;
3: **for all** $o \in O_{kn}$ **do**
4: $s^i_{o_{tar}} \rightarrow s^i_o$, $s^g_{o_{tar}} \rightarrow s^g_o$;
5: Find the functional part $\psi^* \in \Psi_o$ using (5.11);
6: **for all** $\psi \in \Psi_{o_{tar}}$ **do**
7: **if** ψ is similar with ψ^* **then**
8: Add o to O_{src};
9: **end if**
10: **end for**
11: **end for**
12: **if** $|O_{src}| > 1$ **then**
13: Sort O_{src} by the similarity with o_{tar};
14: **end if**

this purpose. The initial state $s^i_{o_{tar}}$ and the goal state $s^g_{o_{tar}}$ of the target object o_{tar} are directly assigned to the corresponding states s^i_o and s^g_o of the known object o (Line 4). Then, a functional part ψ^* can be found for achieving the goal s^g_o using equation (5.11). In other words, the affordance model $e_o = \hat{h}_a(e_\psi)$ is used to find ψ^* that is predicted to result in the closest effect to the desired effect $s^g_o - s^i_o$. The underlying intuition is that the target task could be solved by manipulating the part ψ^* as if o was the target object.

The second step is to decide whether o_{tar} has a part that looks similar with ψ^*. It is anticipated that such a part of o_{tar} is likely to be a functional part for the target task. The measure of part similarity is based on the shape descriptors introduced in Table 5.1. Similar parts are classified into the same group. The k-Nearest Neighbor algorithm or the X-means algorithm can be used to classify the parts. This allows generalization to new parts that have not been seen before. If there exists a part $\psi \in \Psi_{o_{tar}}$ such that ψ looks similar with ψ^*, then o is considered to be a source object, and is added to O_{src} (Line 8).

At this point, if there exists more than one source object, i.e., $|O_{src}| > 1$, then the source objects will be sorted by comparing their similarity with o_{tar} (Line 13). The similarity is measured according to spatial relations between object parts. The reason for using the spatial information is that the position of a part is typically designed for a particular type of use. For example, a top lid is likely to be lifted up or pushed down, while a side lid is likely to be pulled back or pushed forward. The spatial relation between any two parts ψ_i and ψ_j is measured by their relative position in the captured image. In our case, denote by $above(\psi_i, \psi_j)$ the relation "part ψ_i is above part ψ_j". Then, the similarity between two objects is measured by counting such relations between the parts.

5.5.4 How to Transfer

We now discuss the Action Selection and Transfer Evaluation modules in Figure 5.3. Algorithm 6 discusses how the affordances of the selected source objects can be transferred to aid the learning of the target task.

Assume that the robot has already found a set of similar objects O_{src} by Algorithm 5. The most similar object $o_{src} \in O_{src}$ is selected as the source object for transfer learning. The goal state $s^g_{o_{tar}}$ is predefined, and the initial object state $s^i_{o_{tar}}$ is observed by the robot. The aim is to achieve the task goal $s^g_{o_{tar}}$ and find the functional part $\psi^*_{tar} \in \Psi_{o_{tar}}$ along with the action type $a^*_{tar} \in \mathcal{A}$ and the action parameter $\theta^*_{tar} \in \Theta$.

A subset of parts $\Psi_{sim} \subset \Psi_{o_{tar}}$ keeps a record of the similar parts of o_{tar} compared with o_{src}. These parts are likely to include the functional part that can be used to solve the target task. The set Ψ_{sim} is initialized empty in Line 1. Before taking any actions, the robot first carries out a task rehearsal with o_{src} to find its functional part as if o_{src} was being manipulated. The initial state and the goal state of o_{tar} are assigned to the states of o_{src} (Line 2). Then, the functional part ψ^*_{src} is found along with the corresponding action type a^*_{src} and parameter θ^*_{src} (Line 3 and 4). By comparing object features, the potential functional part $\psi^*_{tar} \in \Psi_{o_{tar}}$ is selected and added to Ψ_{sim} (Line 5 and 6). Accordingly, the action type a^*_{tar} and the action parameter θ^*_{tar} are initialized by assigning the values of a^*_{src} and θ^*_{src}.

The robot interacts with the part ψ^*_{tar} using the action type a^*_{tar} from Lines 8 to 16. The action parameter θ_k is initialized in Line 8. This is the result of direct transfer of the source object affordances, i.e., the exploitation mode is turned on. Then, the transfer result will be evaluated by performing the action $a^*_{tar}(\theta_k)$ and checking the goal condition (Line 10 and 11). If the goal is achieved, the transfer learning succeeds and terminates. This is likely to happen when o_{src} has a similar functional part with o_{tar} (see Figure 5.2). However, the required action parameters can vary for similar functional parts. For example, the pushable lids in Figure 5.2(a) and Figure 5.2(c) need different pushing parameters to be opened wide enough. If the goal is not achieved, the robot continues to explore the action parameter space Θ. Based on the collected data $D_{o_{tar},\psi^*_{tar},a^*_{tar}}(k)$, the affordance models (5.7) and (5.8) are learned. Line 13 makes use of the source object's affordance model that provides a gradient direction $\nabla\hat{g}_{o_{src}}(a^*_{tar}, \theta_k)$ for searching effective action parameters. The parameter $\lambda_k > 0$ is a step size that guarantees $\theta_k \in \Theta$. The action parameter space Θ is explored until the maximal number of actions have been performed or the goal condition is satisfied. After the exploration, the exploitation mode is switched on (transfer evaluation). The learned model $\hat{g}_{o_{tar}}(a^*_{tar}, \theta)$ is used to select an action parameter and check whether the goal can be achieved (Line 16).

If the goal is not achieved, Lines 17 through 23 continue interacting with ψ^*_{tar} using other action types. The selected part ψ^*_{tar} is still believed to be a functional part, but the action type a^*_{tar} is not effective any more as a^*_{src} was effective for ψ^*_{src}. For each action type $a \in \mathcal{A} \setminus \{a^*_{tar}\}$, Line 14 selects one action parameter and checks whether the prediction $\hat{g}_{o_{src}}(a, \theta)$ is consistent with the actual observation. Actually, $\hat{g}_{o_{src}}(a, \theta)$ predicts that all the candidate action types in $\mathcal{A} \setminus \{a^*_{tar}\}$ are ineffective. Only when the actual observation is inconsistent with the prediction of $\hat{g}_{o_{src}}(a, \theta)$, the current action type a is likely to be effective for ψ^*_{tar}. For example, the handles in Figure 5.2(b) and Figure 5.2(e) are lift-able and pull-able, respectively. Other action types such as sliding left or right are ineffective

Algorithm 6: Transfer learning of affordances across objects.

inputs A source object $o_{src} \in O_{src}$ and a target object o_{tar};
An initial object state $s^i_{o_{tar}}$ and a goal state $s^g_{o_{tar}}$;
Learned models $e_\psi = \hat{g}_{o_{src}}(a, \theta)$ and $e_{o_{src}} = \hat{h}_a(e_\psi)$, where $\psi \in \Psi_{o_{src}}$, $a \in \mathcal{A}$.
outputs $\psi^*_{tar} \in \Psi_{o_{tar}}$, $a^*_{tar} \in \mathcal{A}$, $\theta^*_{tar} \in \Theta$;

1: Initialize $\Psi_{sim} = \emptyset$;
2: $s^i_{o_{tar}} \rightarrow s^i_{o_{src}}$, $s^g_{o_{tar}} \rightarrow s^g_{o_{src}}$;
3: Select ψ^*_{src} and a^*_{src} using (5.11);
4: Select θ^*_{src} using (5.12);
5: Find $\psi^*_{tar} \in \Psi_{o_{tar}}$ that is similar with ψ^*_{src};
6: Add ψ^*_{tar} to Ψ_{sim};
7: $a^*_{src} \rightarrow a^*_{tar}$, $\theta^*_{src} \rightarrow \theta^*_{tar}$;
8: $k = 1$; $\theta_1 = \theta^*_{tar}$;
9: **while** $k \leq n$ **do**
10: Apply $a^*_{tar}(\theta_k)$ on ψ^*_{tar};
11: Check the goal condition (5.9);
12: Learn (5.7) and (5.8) based on data $D_{o_{tar},\psi^*_{tar},a^*_{tar}}(k)$;
13: $\theta_k \leftarrow \theta_k + \lambda_k \nabla \hat{g}_{o_{src}}(a^*_{tar}, \theta_k)$;
14: $k \leftarrow k + 1$;
15: **end while**
16: Use $\hat{g}_{o_{tar}}(a^*_{tar}, \theta)$ and check the goal condition (5.9);
17: **for all** $a \in \mathcal{A} \setminus \{a^*_{tar}\}$ **do**
18: Select $\tilde{\theta} \in \Theta$, apply $a(\tilde{\theta})$ on ψ^*_{tar} and observe effect;
19: **if** the observation is inconsistent with $\hat{g}_{o_{src}}(a, \theta)$ **then**
20: Collect more data and learn (5.7) and (5.8) for (ψ^*_{tar}, a);
21: Use $\hat{g}_{o_{tar}}(a, \theta)$ and check the goal condition (5.9);
22: **end if**
23: **end for**
24: **for all** $\tilde{\psi}_{src} \in \Psi_{o_{src}} \setminus \{\psi^*_{src}\}$ **do**
25: Find $\tilde{\psi}_{tar} \in \Psi_{o_{tar}}$ that is similar with $\tilde{\psi}_{src}$;
26: Add $\tilde{\psi}_{tar}$ to Ψ_{sim};
27: **for all** $a \in \mathcal{A}$ **do**
28: Repeat Line 17 through 21 substituting ψ^*_{tar} with $\tilde{\psi}_{tar}$;
29: **end for**
30: **end for**
31: Use Algorithm 4 to interact with $\Psi_{o_{tar}} \setminus \Psi_{sim}$.

for both of them. Assume the object in Figure 5.2(b) is the source object, and the object in Figure 5.2(e) is the target object. Each ineffective action type is only needed to be tried once to be confirmed. When the handle is pulled (Figure 5.2(e)), the displacement of the handle is found inconsistent with the prediction. In this case, the pull action is likely to be effective for ψ^*_{tar}. Accordingly, more pull parameters are selected to learn the affordance model (Line 20). Then, the learned model $\hat{g}_{o_{tar}}(a, \theta)$ is used to select an action parameter and check whether the goal can be achieved (Line 21).

Till Line 23, the part $\psi^*_{tar} \in \Psi_{o_{tar}}$ has been proved nonfunctional. Lines 24 through 29 continue to check other similar parts between o_{src} and o_{tar}. These parts are predicted to be nonfunctional, and the prediction needs to be verified by the robot. The action selection for each part in $\Psi_{sim} \setminus \{\psi^*_{src}\}$ is done in the same way as Line 18. Finally, Line 31 uses Algorithm 4 to interact with the other parts $\Psi_{o_{tar}} \setminus \Psi_{sim}$ that seem to be novel compared with o_{src}.

We note that the robot makes full use of the affordance models of the source object. In other words, it not only uses the successful experience with the functional part (Line 3 to 15), but also uses the unsuccessful experience with the functional part (Line 19) as well as the non-functional parts (Line 25 to 29).

5.6 Experiments

We still use the humanoid robot NAO in an actual household task. We focus on transfer learning of affordances in order to open different types of lids. We assume that all the lids are openable for the NAO.

5.6.1 Task Setting

In our experiment, we used the five garbage cans in Figure 5.2 to test the part-based affordance model and the transfer learning architecture. The set of objects was denoted by $O = \{o_1, o_2, o_3, o_4, o_5\}$, in which o_1 and o_3 had pushable lids (Figure 5.2(a) and Figure 5.2(c)), o_2 and o_4 had liftable lids (Figure 5.2(b) and Figure 5.2(d)), and o_5 had a pullable lid (Figure 5.2(e)). They were presented to NAO in sequence, and NAO learned about one garbage can at a time. In each learning task, a garbage can was positioned approximately 10 to 12 cm in front of NAO and the area to be explored was about 25 to 45 cm high. These values agreed with the capabilities of NAO due to its height and the length of its arms. The left arm of NAO was used to interact with the garbage cans.

The bottom camera on NAO's head was used as the main sensory input (640×480 resolution). For each garbage can, a blue marker ($5\,\text{cm} \times 2\,\text{cm}$) was used for the recognition of lid (with a NAO marker at its center), and a red marker ($5\,\text{cm} \times 2\,\text{cm}$) for the recognition of the garbage can body, and a green marker for the recognition of the handle ($10\,\text{cm} \times 3\,\text{cm} \times 1\,\text{cm}$), if there was one. The camera images were transformed into HSV color space. Color blobs and the bounding boxes were obtained using the OpenCV library[1]. As a result, object parts were denoted by $\Psi_{o_i} = \{\psi_{l_i}, \psi_{b_i}\}, i = 1, 3$, and $\Psi_{o_i} = \{\psi_{l_i}, \psi_{b_i}, \psi_{h_i}\}, i = 2, 4, 5$ where ψ_{l_i} denoted a lid, ψ_{h_i} denoted a handle, and ψ_{b_i} denoted a body. For each $\psi \in \Psi_{o_i}$, its

[1] http://opencv.org/

state $s_\psi = (x_\psi, y_\psi)$ was described by the 2D coordinates of the bounding box center, based on which the spatial relation between two parts were obtained.

The set of action types for learning affordances was $\mathcal{A} = \{a_1, a_2, a_3, a_4, a_5, a_6\}$, which were {*sweep left*, *sweep right*, *push forward*, *pull back*, *lift up*, *sweep down*}. Each action type $a_i \in \mathcal{A}$ was constrained by the action parameter $\theta \in \Theta \subseteq \mathbb{R}^3$. For example, in the Cartesian space of NAO, a push action was defined in the parameter space $\Theta = \{\mathbb{R}^3 | 0 < x < 0.10, y = 0, z = 0\}$ (in meters).

To detect the opened area, we put a black plastic bag in each garbage can and calculated the area of the dark part in a captured image. The opened area was also located by a bounding box with a size of $w \times h$ in pixels. Then, s_o was the absolute value of opened width in meters:

$$s_o = \alpha h \tag{5.14}$$

where α was used to normalize h according to the relative size of known markers. In all experiments, initial object states were the same $s_o^i = 0$ when the lids were tightly closed, and the goal states were also the same $s_o^g = 0.1$ m.

After an action was performed on ψ, the new states $s'_\psi = (x'_\psi, y'_\psi)$ and s'_o were extracted from a new image. The displacement of e_ψ was calculated from (5.1)

$$e_\psi = \alpha \sqrt{(x'_\psi - x_\psi)^2 + (y'_\psi - y_\psi)^2} \tag{5.15}$$

For this learning problem, we chose a linear regression model to approximate g_o and h_a:

$$\hat{g}_{o_j}(a_k, \theta) = b_{j,k} + c_{j,k}\,\theta \tag{5.16}$$

$$\hat{h}_{a_k}(e_\psi) = d_{j,k} + e_{j,k}\,e_\psi \tag{5.17}$$

The reason for this choice was due to the character of the functional parts operation, i.e. pushing or lifting further results in more opening.

The parameters in (5.16) were estimated by minimizing the residual sum of squares (RSS) using m observation samples:

$$S_m = \Sigma_{l=1}^{m} (e_l - \hat{g}_{o_j}(a_k, \theta_l))^2 \tag{5.18}$$

We obtained the parameters of (5.17) in the same way. We set $n = 5$ in Algorithm 4 and 6. The regressions (5.16) and (5.17) were considered accurate when the following condition was met:

$$|S_{m+1} - S_m| < \delta_3 \tag{5.19}$$

where δ_3 was a small positive real value.

5.6.2 Results

As a baseline, we first ran Algorithm 4 for five garbage cans without transfer learning. Then, we used the learned affordances to evaluate the transfer results. In total, 156 regression models were learned in the form of (5.16) and (5.17). Some of the results are shown in Table 5.3.

Table 5.3: Learned affordance models with the five objects.

O	ψ	a	$\hat{g}_{o_i}(a,\theta)$	$\hat{h}_a(e_\psi)$
o_1	ψ_{l_1}	**push**	$-0.004 + 0.574\,\theta$	$0.008 + 1.833\,e_\psi$
o_2	ψ_{h_2}	push	$0.588\,\theta$	$0.003 + 0.663\,e_\psi$
o_2	ψ_{h_2}	pull	$-0.001 + 0.452\,\theta$	$0.006 + 0.475\,e_\psi$
o_2	ψ_{h_2}	**lift**	$-0.013 + 1.866\,\theta$	$0.010 + 0.508\,e_\psi$
o_3	ψ_{b_3}	push	$0.659\,\theta$	$0.003 + 0.364\,e_\psi$
o_3	ψ_{l_3}	**push**	$-0.001 + 0.642\,\theta$	$0.013 + 2.014\,e_\psi$
o_4	ψ_{h_4}	push	$0.001 + 0.069\,\theta$	$0.006 + 1.435\,e_\psi$
o_4	ψ_{h_4}	**lift**	$-0.009 + 2.891\,\theta$	$0.003 + 0.590\,e_\psi$
o_5	ψ_{h_5}	sweep	$-0.002 + 3.205\,\theta$	$0.009 + 1.314\,e_\psi$
o_5	ψ_{h_5}	**pull**	$0.001 + 0.240\,\theta$	$0.017 + 1.923\,e_\psi$
o_5	ψ_{h_5}	lift	$0.522\,\theta$	$0.011 + 1.324\,e_\psi$
o_5	ψ_{l_5}	push	$0.006 + 1.070\,\theta$	0

The five functional parts were all found out correctly using the learned models. Also, effective action types (marked in bold) and associated parameters were selected to achieve the task goal. Due to sensory noise and the design of garbage cans, some pairs of part and action type also resulted in the observation of opening, e.g., (ψ_{h_4}, pull) and (ψ_{b_3}, push). However, the selection of action parameter was out of the allowed action parameter range or the predicted opening size could not satisfy the goal condition.

Table 5.4 illustrates the number of required exploratory actions with affordance transfer (N_{tran}) and without transfer ($N_{o_{tar}}$), together with the chosen source objects (o_{src}), if there was one.

Table 5.4: Comparison of the required number of actions for learning target object o_{tar} without transfer ($N_{o_{tar}}$) and with transfer (N_{tran}) from a source object o_{src}.

o_{tar}	$N_{o_{tar}}$	N_{tran}	o_{src}
o_1	36	-	-
o_2	65	29	o_1
o_3	53	1	o_1
o_4	83	3	o_2
o_5	80	11	o_2

Without Transfer Learning

NAO typically interacted with every object part with all 6 action types. If an action type resulted in no significant displacement of a part, this action would be considered ineffective. For example, sweeping left over a pushable lid for 3 times was enough to learn an accurate prediction model that this action was ineffective for this type of lid. It required 36 exploratory actions, which was less than executing each action type for $n = 5$ times, i.e., collecting $2 \times 6 \times 5 = 60$ samples for o_1.

Transfer Learning

The most difficult case was learning o_2 when o_1 was selected as the source object. NAO started with pushing the lid, which was found ineffective. Thereafter, other action types were tried on the lid to check whether they were still ineffective as anticipated. Then, the robot found o_1 and o_2 shared a similar body. checked the body part of o_1 with a small number of actions (Lines 24 through 29 in Algorithm 6). Finally, the robot interacted with the handle which was not seen before. In total, it took 29 exploratory actions, still fewer than 65 without transfer.

The most efficient transfer learning happened for learning o_3 and o_4 after o_1 and o_2 had been learned. Because o_1 and o_3 had the similar pushable lid which was above the body part, while o_2 and o_4 had the similar liftable handle as well as the same spatial relations. In the case of learning o_3, object o_1 was selected as the source object because o_1 was more similar with o_3 than o_2. The lid (ψ_{l_3}) and the "push" action (a_3) were selected, with a parameter 0.087 applied, which resulted in the state of opening $s'_o = 0.14 > s^g_o$. The goal state was achieved with only one single action. In the case of learning o_4, object o_2 was selected as the learning source because it was more similar than o_1 and o_3. NAO tried the lift action (a_5) on the handle (ψ_{h_4}) to achieve the desired opening, each time with a bigger lift parameter given by Line 13 in Algorithm 6, where $\nabla\hat{g}_{o_2}(a_5, \theta) = 1.866$ (see Table 5.3). This parameter selection policy suggested that lifting ψ_{h_4} more would result in more opening.

However, lifting the handle did not work for o_5 when o_2 was selected as the learning source. Then, NAO tried action types $a \in \mathcal{A} \setminus \{a_5\}$ (Lines 17 through 23 in Algorithm 6). It happened to find that pulling (a_4) the handle generated a different effect than model $\hat{g}_{o_2}(a_4, \theta)$ predicted. Therefore, it tried several more pulling parameters to learn an accurate prediction model $\hat{g}_{o_4}(a_4, \theta)$. The goal condition was satisfied in Line 17 and the learning was terminated.

5.7 Conclusions and Open Issues

In this chapter, we have investigated a transfer learning approach to handle a new object with additional affordance knowledge of known objects. We have developed a part-based model to represent the affordances of complex household objects composed of several parts. Each part is associated with a set of affordance models learned through performing the corresponding set of parameterized actions. Based on the predictions of the learned models, a pair of part and action can be selected to achieve a desired effect. When the new object is given in a goal-directed task, the potential functional part can be found by measuring its similarity with the functional parts of known objects. The associated action parameters can also be found to achieve the goal.

An experimental evaluation of the proposed method has been carried out in a lid opening task. The state space and the action space are both one-dimensional continuous spaces. Linear regression models have been used to approximate the affordance models. In the baseline experiments, the robot has to learn from scratch without knowledge transfer. It interacts with every object part to learn the associated affordance models, and then uses the models to find the functional part along with the action parameter. In contrast, our trans-

fer learning approach always manages to directly reuse the affordance models of relevant known objects, not necessarily having to try every object part. Compared with the baseline, transfer learning has largely reduced the number of actions that are needed to achieve the lid opening task. This indicates that transfer learning is more efficient than learning from scratch because it avoids unnecessary exploratory actions whenever possible. Specifically, transfer learning is most efficient when the new object has a more similar way of manipulation with a known object. However, transfer learning can suggest to select inappropriate actions when the new object looks similar to a known object, yet has a different way of manipulation. It is interesting to investigate how to improve the performance in case of such a negative transfer [120].

The linear regression models have shown good performance in the lid opening task under the assumption that the lid was initially closed. In chapter 4, we have used neural networks to model affordances considering any possible initial state of a lid, and we have shown that affordance models are not necessarily linear. Such a non-linear model would be more appropriate than the linear regression models for describing the movement patterns of the functional parts of general household objects. However, our focus in this chapter has been to show the effectiveness of the transfer learning approach. In principle, there is no obstacle to use non-linear models to represent and transfer affordances. It is an immediate extension to the current research to use non-linear regression models to deal with higher dimensional continuous state and action spaces. Active learning discussed in chapter 4 could be combined with transfer learning to further speed up the exploration in the action spaces.

We have assumed that the goal states are exactly the same in this chapter, i.e., the goals are opening the lids wide enough for all garbage cans. It will be useful to also consider transfer learning with different goals. For instance, the experience of opening a lid could be helpful for learning how to close a door. Also, it is interesting to consider transfer learning with a sequence of goals with progressively increasing complexity. Methods to do these have been developed in the multi-task reinforcement learning (MTRL) framework [129] in which the agent needs to solve a sequence of Markov Decision Processes (MDPs) chosen randomly from an unknown distribution. Techniques that reuse the previously learned distribution over MDPs as an informed prior, such as hierarchical Bayesian MTRL [130], are a promising approach.

This chapter has considered the manipulation of a single object part at a time. We have separately modeled the movement of the part and the part-object relation. Consequently, it is possible to first find the functional part, and then the action parameters to achieve the given goal. However, we have not considered the relations between multiple parts. For example, lift/pull the handle in Figure 5.2 not only moves the handle, but also moves the lid. In the future, it is interesting to investigate the co-movement of object parts and discover their underlying relations. It will be useful to also consider more complex objects with more than one functional part.

CHAPTER SIX

Integration of Affordance Learning and Symbolic Reasoning

This chapter introduces a robot control architecture that subsumes the three architectures proposed in previous chapters. This architecture facilitates affordance learning and reasoning at different cognitive levels. As discussed in the previous chapters, affordance learning takes place at the sub-symbolic level through embodied robot interaction with environments. In contrast, reasoning takes place at a higher, symbolic level. The proposed cognitive control architecture integrates these two levels so that they can effectively interact. Specifically, we use the agent programming language GOAL to program the cognitive layer which supports symbolic knowledge representation and high-level action selection. The knowledge base includes task domain information as well as affordances.The action selection is based on the knowledge base, task goals and the world states. After performing the selected actions and observing the actual effects, affordances are obtained and sent back to the cognitive layer to update the knowledge base. In this way, the affordance knowledge in the cognitive layer is grounded in the robot's own sensory-motor learning experience, while affordance learning is controlled by the cognitive layer. The proposed architecture is a step towards long-term affordance learning and enhances the robot's ability to solve complex real world tasks. We demonstrate how the architecture works through a garbage disposal task with the humanoid robot NAO.

6.1 Introduction

The previous chapters of this thesis have focused on the sensory-motor level robot interaction with environments. Affordances can be learned and used for action selection to achieve a desired goal. However, these approaches suffer from two limitations that restrict the learning and use of affordances in more complex tasks. First, reasoning is not supported. This hinders the use of affordances for action selection in multi-step tasks and uncertain task environments. Second, we have assumed so far that the robot is always in learning mode. Because affordance learning involves exploration in the action space of the robot, it can be very time consuming to execute actions and collect data needed for learning. It would be better if the robot could decide itself when to initiate affordance learning, and when to

terminate the learning. To address these limitations, this chapter proposes a cognitive robot control architecture that facilitates sophisticated reasoning as well as efficient affordance learning. Specifically, we provide solutions for the following problems:

- *How can affordances be used by a cognitive agent to select actions based on symbolic reasoning?* In the previous chapters, affordances are represented at the sub-symbolic level, and they are directly learned from sensory-motor data. Task goals are represented in the perceptual space, and then affordances are used to select low-level motor actions to achieve the goals. However, this approach of affordance learning and use provides limited support for symbolic reasoning, e.g., [131, 132]. Such reasoning is based on symbolic representation of goals and task domain knowledge. In this chapter, we represent affordances as symbolic knowledge. We use an environment interface to obtain a symbolic representation of an affordance from a non-symbolic representation. Then, the symbolic affordance knowledge is taken into account by the action selection mechanism that is based on symbolic reasoning.

- *How can reliable affordances be maintained for solving tasks effectively?* The reliability of an affordance should be sufficiently high for effective task execution. Otherwise, the robot would easily fail to achieve desired effects by using the affordance. In other words, affordances should not be used unless the effects of actions on objects have been established to a sufficient degree. Besides, affordances may change in dynamic environments. We need to update the reliability of affordances based on the long-term sensory-motor experience of the robot.

- *How can a robot decide to switch between an exploration mode (learn affordances) and an exploitation mode (use affordances to achieve a goal)?* It is a challenge for the robot to take control over the affordance learning process during task execution. There is no need for the robot to explore the sensory-motor space if the robot already has the relevant knowledge to solve a task. We need to identify the relevant conditions that settle the question whether a robot should continue to learn or not.

In this chapter, we propose a cognitive robot control architecture that combines reasoning at the symbolic level and affordance learning at the sub-symbolic level. We take the approach of agent-oriented programming that supports the programming of cognitive agents. Specifically, we use the agent programming language GOAL[1] [133] to program the cognitive agents. A GOAL agent maintains its goals, beliefs and knowledge in a symbolic form for high-level action selection and reasoning. Using the Environment Interface Standard (EIS) [134], the agent in the cognitive layer can communicate with lower sub-symbolic layers that are responsible for robot behavior control. The agent maintains symbolic affordance knowledge that can be created in a top-down manner [39, 40] or learned from the sensory-motor experience in a bottom-up manner [33, 135]. The symbolic affordance knowledge can be used by the agent for action selection through affordance-aware reasoning. In addition, the reliability of the affordance knowledge is updated to reflect its long-term success for achieving desired effects. Finally, conditions are introduced for the agent to activate and terminate the learning of affordances.

[1] https://github.com/goalhub

The remainder of this chapter is structured as follows. Section 6.2 briefly discusses the related work. Section 6.3 introduces the cognitive robot control architecture. Section 6.4 describes the experiments and evaluation. Finally, Section 6.5 concludes the chapter.

6.2 Related Work

In the literature, several cognitive architectures have been proposed such as SOAR [136, 137] and CRAM [138]. Both SOAR and CRAM are similar to the BDI-based [139] cognitive architecture that our approach is based on. Our contribution is the integration of affordance learning into the cognitive architecture. In our architecture, not only the learning of affordances is allowed for handling novel objects and developing novel skills, but also affordances can be used for sophisticated reasoning and action selection.

Based on bottom-up robot learning and use of affordances, affordance-aware reasoning has been addressed at the sub-symbolic level. In [33], affordances are learned from sensory-motor experience of the robot, and then they are used for multi-step action planning through forward chaining[2]. Similarly, based on the learned affordances, a one-step action is selected with the highest probability to achieve the demonstrated goal in an imitation task [10]. Chapter 3, 4 and 5 have also followed this path of affordance learning and use at the sub-symbolic level. However, these approaches have not represented affordances at the symbolic level, which limits the use of powerful artificial intelligence (AI) techniques for symbolic reasoning with affordances. In this chapter, cognitive agents maintain affordance knowledge at the symbolic level, and these agents support sophisticated symbolic reasoning by using the affordance knowledge.

Different with the bottom-up learning and use of affordances at the sub-symbolic level, functional affordances have been used for reasoning only at the symbolic level. In [131], object functionalities are considered to find substitute objects to achieve task goals, e.g., using a mug instead of a glass for water service. In [41], affordance template (AT) is proposed to represent objects (e.g., turnable wheels and climbable ladders) for efficient teleoperation of robots by human operators. However, these approaches currently do not support affordance learning. Similar with [131, 41], the approach of [132] has assumed the use of high-level action rules without considering low-level robot control problems. In other words, the high-level affordance knowledge cannot be verified by the sensory-motor experience of a real robot. In contrast, our architecture supports the verification of affordance knowledge through embodied robot interaction with objects in real world environments.

Our work is closely related to the research on object-action complexes (OACs) [59]. We share the same motivation to bridge the gap between low-level robot control problems and high-level symbolic reasoning through affordance learning and use. We also both take into account the verification of symbolic affordance knowledge. An OAC captures the interaction between an object and the robot in terms of: 1) an action execution specification, 2) a prediction function about state changes caused by the execution of robot actions, and

[2]Forward chaining uses a tree structure with nodes representing the perceptual states and edges corresponding to object-action pairs. An object-action sequence is found to transform an initial state into a goal state in the continuous perceptual state space [33].

3) a statistical measure of the success of the OAC. Similarly, the robot actions in our architecture are executed in a low-level control space, and our affordance model predicts action effects as well. We also maintain long-term success/failure statistics about the affordance knowledge. However, the OAC approach separates the learning and the use of affordances. As a result, affordances are learned off-line and they are not verified on-line during task execution. In contrast, we use autonomous agents to take control of affordance learning and use. Affordance learning can be initiated whenever the learning is required during task execution.

The recent work [135] is also related to our research. It proposes bottom-up learning of affordance-based logical rules. Symbols and operators are learned from low-level exploitative manipulation experience of the robot. The representation of object categories is based on unsupervised clustering of action effects. Then, these discrete categories are used to generate logical rules that make predictions of action effects associated with object features. However, this approach does not support the on-line learning and verification of affordances. In addition, unlike our architecture, the bottom-up approach does not allow humans to provide symbolic affordance knowledge as done in [41, 131, 132].

The main contribution of this chapter is that we propose an agent-based robot control architecture that facilitates both affordance-aware reasoning at the symbolic level and affordance learning at the sub-symbolic level. In this chapter, a cognitive agent maintains affordances as symbolic knowledge that is not only used for symbolic reasoning, but also is updated by the sensory-motor experience of the robot during on-line task execution. In addition, the learning of affordances is controlled by the agent without human intervention. These key features distinguish our cognitive robot control architecture from other approaches in the literature.

6.3 Cognitive Affordance Learning Architecture

This section introduces our cognitive robot control architecture. We discuss several issues including how the sensory data and action commands are processed (section 6.3.1), how symbolic affordance knowledge is used for action selection (section 6.3.2), how the update of affordance knowledge is carried out (section 6.3.3), and how affordance learning is controlled by the architecture (section 6.3.4).

6.3.1 Architecture Overview

The main components of our architecture are illustrated in Figure 6.1, which includes a symbolic layer for cognitive control, and a sub-symbolic layer for embodied-robot behavior control.

Agent-based Cognitive Control

The symbolic layer provides support for symbolic reasoning and high-level decision making. It functions as a task manager for the robot. In our architecture, we choose the *goal-oriented agent language* (GOAL) [133] to program cognitive agents for robot control. A GOAL agent uses a symbolic, logical language (e.g., Prolog [140]) to represent information

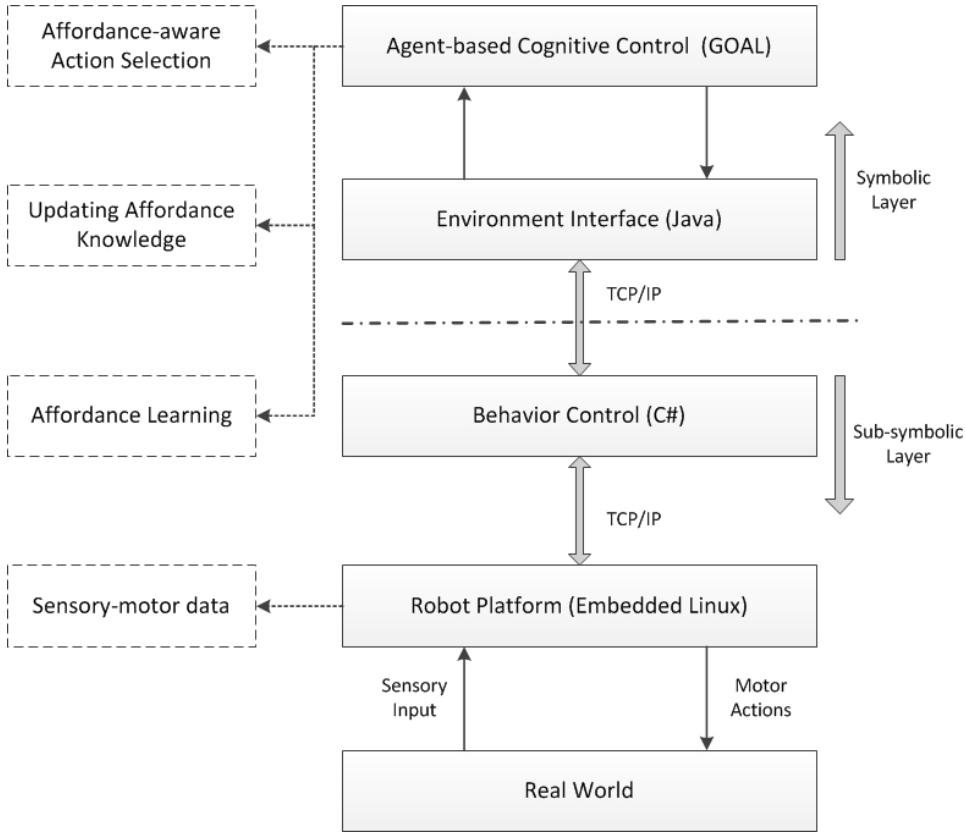


Figure 6.1: The overall design of the architecture.

about the environment. Based on such information, high-level action choices are derived to achieve task goals. Refer to Appendix B for details about GOAL agents.

First of all, we use a symbolic representation of affordance knowledge, and we maintain the statistics of the affordance knowledge. The statistics include the counts of times that an affordance has been used and proved successful, along with the reliability of the affordance knowledge calculated from these counts (see section 6.3.2). Based on this representation, the agent can use the affordance knowledge to select actions. Whenever an affordance is used for action selection, the actual action effect is observed and the statistics of the affordance are updated (see section 6.3.3). In our architecture, affordance learning is controlled by a GOAL agent that makes high-level decisions to switch on/off the learning. We introduce decision rules for this purpose. For example, if there exists reliable affordance knowledge for action selection, then the agents exploits the knowledge and affordance learning is unnecessary. Otherwise, the learning is switched on (see section 6.3.4).

Environment Interface

The interaction between the symbolic and sub-symbolic layers is managed by the environment interface standard (EIS) [134] implemented in Java. As these layers may use different programming languages to represent symbolic and sub-symbolic information, an interface is needed to translate between the symbolic and sub-symbolic representations. For this purpose, EIS acts as a bridge between high-level cognitive control and low-level behavior control. It deals not only with symbolic representation required for logical reasoning in the cognitive layer, but also with the translation of high-level decisions into low-level control programs. Moreover, sensory data can be mapped into a symbolic form which can be used by cognitive agents.

The interaction between the agent and the environment interface is twofold. First, the agent sends action commands to the environment interface for execution. A robot is assumed to have a sufficient repertoire of motor skills to carry out manipulation tasks such as reaching, grasping, pushing, lifting, pulling, and so on. Each motor skill is associated with an action defined in the action specification section of an agent (see Appendix B). In this way, an action of the agent can be executed by the robot in the real world environment. Second, the environment interface sends back new information to update the belief base of the agent. This involves the transformation from low-level raw sensory data to high-level symbolic representation. The sensory data is processed into a feature vector which is described in an *attribute space* (see section 3.2.1). Then, a message describing this feature vector is sent to the environment interface, where the feature vector is mapped into a label. In this way, a symbolic percept is derived from the label and used to update the belief base of the agent. Such a discrete representation of objects is convenient for reasoning at the symbolic level.

Behavior Control

The sub-symbolic layer is responsible for robot behavior control at the sensory-motor level. A behavior is the executed perception-action routine that enables embodied robot interaction with its environment [9]. In order to develop robot behaviors, sensory functions have to be prepared such as object recognition, object tracking, self-localization, etc. At the

same time, motor skills should be available such as controlling the end-effectors in Cartesian spaces. These sensory functions and motor skills are developed using available APIs, in our case, within the .NET framework (C#). These functions and skills are also basic requirements for affordance learning, as discussed in previous chapters. We note that the learning of affordances can be seen as a behavior as well because it involves sensory-motor coordination in a perception-action loop.

Robot Platform

We use a humanoid robot NAO that runs an embedded Linux system. It is remotely connected to a separate computer via Wi-Fi or Ethernet. The NAO collects raw sensory data and sends the data to the computer, where the data is processed and action commands are sent back to the NAO. Refer to section 3.4 for more details.

6.3.2 Affordance-aware Action Selection

Affordance-aware action selection makes use of affordance knowledge for effective and efficient achievement of task goals.

Representation of Affordance Knowledge

In previous chapters, we have modeled an affordance as the relation between *Object*, *Action*, and *Effect* (see **Definition 3.1**). The affordance formalism concerns the coupling of object features and robot actions for achieving desired effects. *Object* encodes sensory information about object attributes (e.g., color, shape, size and location), and *Effect* is a concept based on object state changes. As it is in general unknown what effects would be achieved before the robot actually interacts with objects, we should not hard code affordances as static and unchanged facts in the *knowledge base* of a GOAL agent. Instead, we represent affordances in the *belief base* of the agent, because facts that may change at runtime should be stored there (see Appendix B).

In this chapter, we extend the concept of an affordance in **Definition 3.1** with a record of the statistics of the results of applying the affordance. This allows the agent to make reasonable decisions when using affordances for action selection. Affordance knowledge in the belief base of a GOAL agent is represented as follows:

```
afford(Obj,Act,Eff,Tot,Suc,Rel)
```
(6.1)

where `afford` specifies that this belief describes affordance knowledge, `Obj` denotes an object, `Act` denotes an action, and `Eff` denotes the consequent effect. In addition, the variable `Tot` counts the total times that the affordance has been used, `Suc` counts the times of successful results of using the affordance, and `Rel` is the reliability of the affordance. The calculation of `Rel` is based on `Tot` and `Suc`. Each affordance is initiated with a reliability value, and then the reliability is updated according to the sensory-motor experience of the robot. The details of how to update the reliability will be discussed in section 6.3.3. In our garbage can manipulation example (see Figure 5.2), `afford(lid,push,opened,`$2, 1, 0.5$`)` means the agent believes that the `lid` can be opened by a `push` action, and the `lid` has been

pushed for two times, one of which results in the `opened` effect. The reliability of the affordance is 0.5.

The representation (6.1) is similar to the representation of object-action complexes (OACs) [59] that maintains a statistical measure of the success of how a robot action can change an object state. Both OACs and our approach consider symbolic reasoning as well as sub-symbolic control problems, unlike Affordance Templates [41] that only consider sub-symbolic robot control or high-level action rules [132] that only consider symbolic reasoning. Compared with the OACs, our approach has a clearer separation between the symbolic layer and the sub-symbolic layer (see Figure 6.1). In addition, the reliability of affordance knowledge is updated and used in a more sophisticated way for action selection.

Action Selection

Following the rule-based action selection mechanism in GOAL (see Appendix B), we take into account affordances when programming action rules. Such an affordance-aware action rule is of the form

```
if bel(obs(Obj)), goal(des(Obj,Eff)),
   bel(afford(Obj,Act,Eff,Tot,Suc,Rel),Rel≥ τ)
then select(Act).
```
(6.2)

where the mental state condition consists of the following elements:

- an object `Obj` that is observed and being manipulated, e.g., `handle`, `lid`, `wheel`, etc.
- a goal `des(Obj,Eff)` which describes the desired effect `Eff` on the object `Obj`.
- available affordance knowledge `afford(Obj,Act,Eff,Tot,Suc,Rel)` which suggests the effect `Eff` can be achieved by performing `Act` on `Obj` with a reliability `Rel`.
- a condition Rel$\geq \tau$ that makes sure an affordance is reliable enough to be used for action selection, where $0 < \tau < 1$ is a given threshold.

When an object `Obj` is observed, the agent inspects the mental state condition to check whether there exists relevant affordance knowledge to achieve the goal `Eff`. If the reliability is high enough (`Rel` $\geq \tau$), the agent predicts that the corresponding action `Act` is still effective to achieve the goal, therefore the action `Act` is selected for execution. The higher the value of τ, the stricter the condition is to use the affordance. We encourage the agent to use the affordance knowledge, meanwhile the threshold should not be too low. Therefore, we instantiate $\tau = 0.6$ by default.

Assume there is an affordance `afford(lid,push,opened,2,2,1)`, which suggests push open a lid has been tried twice and verified twice. In the case of Figure 5.2(a), the agent observes the `lid`, and it believes the action `push` is reliable enough to achieve the `opened` effect. Therefore, the agent selects the action `push`.

We note that the reliability of affordances should be compared for action selection when there are several affordances available. These affordances make different predictions about action effects, and suggest different actions for selection. The agent must choose from the

affordances and select an action for execution. We propose that the agent choose the affordance with highest reliability, which is a reasonable choice. In case that there are two affordances `afford(handle,pull,opened,`$5, 4, 0.8$`))` and `afford(handle,lift,opened,` $2, 2, 1$`))`, the agent would select the action `lift` rather than `pull`.

If the reliability of the affordance knowledge is not high enough (`Rel`$< \tau$), the agent will not use the affordance knowledge for action selection. In this case, affordance learning is necessary. We will discuss the details of affordance learning in section 6.3.4.

6.3.3 Updating Affordance Knowledge

In order to maintain useful symbolic affordance knowledge, the agent's beliefs of the affordance knowledge should be updated through embodied robot interaction with the real world. The importance of verifying symbolic knowledge for embodied robots has been addressed in the literature [4, 59].

In the belief base of a GOAL agent, affordance knowledge can either be created in a top-down manner, or learned bottom-up by the robot itself. In the former case, the agent creates a initial set of affordance knowledge before the learning starts. For example, `afford(lid,push,opened,`$0, 0, 0.1$`))` suggests that a lid might be opened by a push action, the reliability is $0.1 < \tau$ and this affordance has not been verified yet. In the latter case, affordance knowledge is learned from the sensory-motor data, sent to the agent via EIS.

As discussed in section 6.3.2, the reliability of an affordance is important for affordance-aware action selection. In the sequel, we explain how the reliability is updated through information flow between the symbolic layer and the sub-symbolic layer in our proposed cognitive control architecture (see Figure 6.1). The process takes the following three steps:

- **Action Selection**: By using affordance knowledge, a cognitive agent selects an action that is likely to result in a goal, as introduced in section 6.3.2.
- **Action Execution**: The selected action that is expected to achieve the goal is sent through the environment interface to the behavior control module for execution.
- **Effect Verification**: After the action is performed, the actual effect is observed. Then, a new percept is sent back to the agent through the environment interface, and the statistics of `Tot`, `Suc` and `Rel` are updated.

In the action selection step, the agent believes that the selected affordance knowledge ϕ = `afford(Obj,Act,Eff,Tot,Suc,Rel)` correctly predicts the action effect. Such a prediction is going to be verified through executing exactly the same action on the object. In the verification step, the actual effect `Enew` is obtained in the sub-symbolic layer. The agent decides whether the affordance knowledge ϕ is verified or falsified by comparing `Eff` and `Enew`. If the effect `Enew` matches with the predicted effect `Eff`, the latest verified affordance knowledge ϕ' = `afford(Obj,Act,Eff,1,1,1)` is added to the database of percepts via EIS. Otherwise, ϕ' = `afford(Obj,Act,Eff,1,0,0)` is added. Then, ϕ and ϕ' are merged, and the new reliability $\texttt{Rel}_{new}$ is calculated as follows:

$$\texttt{Rel}_{new} = \texttt{Rel}_{\phi} + (\texttt{Rel}_{\phi'} - \texttt{Rel}_{\phi})\frac{1}{\texttt{Tot} + 1} \tag{6.3}$$

where $\texttt{Rel}_{\phi'}$ is the reliability of ϕ'. It means that the new reliability is updated towards $\texttt{Rel}_{\phi'}$. If $\texttt{Rel}_{\phi'} > \texttt{Rel}$, the reliability increases; otherwise, the reliability decreases. The update strategy gives more weight to the earlier results when `Tot` is small. It encourages an early success by quickly raising the reliability above the threshold τ if an effective action has been found. Specifically, this approach is efficient for verifying preprogrammed affordances that are likely to be true but have not been verified yet.

We note that the variables `Eff` and `Enew` both take discrete values such as `opened`, `unopened`, `moved`, `unmoved`, etc. The discretization of effects is conducted in the environment interface. The categories of effects can be predefined as labels [55] or can be obtained by robot in a bottom-up manner [135].

6.3.4 Switching On/Off Affordance Learning

In goal-directed tasks, a robot is expected to achieve task goals as soon as possible. Affordance learning is not always needed because it takes time for the robot to interact with objects and collect data for the learning. If the robot already has acquired reliable enough affordance knowledge for object manipulation, it is more efficient to exploit its knowledge than to spend time on the learning. Therefore, we propose that affordance learning should be carried out under certain conditions.

Learning From Scratch

In case that no reliable affordance knowledge is available, affordance learning can be initiated by the GOAL agent. This happens when a given object is a novel one that has never been manipulated before. The following rule

if `goal(des(Obj,Eff)),bel(obs(Obj),afford(Obj,Act,Eff,Tot,Suc,Rel),Rel<`τ`)`
then `learn(Eff,Obj,mode)`. (6.4)

indicates that if the agent believes there is no reliable affordance knowledge (Rel<τ) about the observed object `Obj`, then affordance learning is required. The learning command `learn(Eff,Obj,mode)` consists of the behavior name `learn`, the goal `des(Obj,Eff)`, the current object `Obj` being manipulated, and an action exploration mode `mode` that indicates how to explore in the continuous parameter space of an action type. For example, random exploration mode, active learning mode (Chapter 4) or transfer learning mode (Chapter 5) can be chosen as the learning mode.

The agent has to select a robot action $a \in \mathcal{A}$ to learn affordances. First of all, the agent initiates a set of unverified affordances in the form of:

$$\texttt{afford(Obj,Act,Eff,}0, 0, r_0\texttt{)} \tag{6.5}$$

where $\texttt{Act} \in \mathcal{A}$, and $r_0 < \tau$ is a small positive real number so that the condition of the rule (6.4) can be satisfied to initiate affordance learning. For example, we instantiate $r_0 = 0.1$ in our experiments. Similar to the *Roulette Wheel Selection* used for action selection in XCS [85] (see chapter 3), the agent selects an action `Act` with a probability proportional to the reliability of the corresponding affordance. Then, affordance learning continues by exploring in the parameter space of `Act`.

In accordance with Chapter 4 and 5, we focus on learning affordance models that predict action effects in continuous action spaces. In this chapter, a prediction model H maps from Θ to the effect space $\mathcal{E}$:

$$H : \Theta \rightarrow \mathcal{E} \tag{6.6}$$

We note that the mapping (6.6) can be seen as the combination of the mappings (5.4) and (5.5). The prediction model H can be learned as:

$$e = \hat{H}(\theta) \tag{6.7}$$

where $e \in \mathcal{E}$, $\theta \in \Theta$, and $\hat{H}$ is the approximation of H. We use linear regression to obtain $\hat{H}$. The maximal effect e_{max} is calculated as follows:

$$e_{max} = \max_{\theta \in \Theta} \hat{H}(\theta) \tag{6.8}$$

The desired effect `Eff` is achieved when the following condition is satisfied:

$$e \geq \delta \tag{6.9}$$

where δ is a threshold. Combining equations (6.8) and (6.9), the effect `Eff` is predicted to be true under the following condition:

$$e_{max} \geq \delta \tag{6.10}$$

Then, the following message

$$\texttt{afford(Obj,Act,Eff,}0, 0, r_1\texttt{)} \tag{6.11}$$

is sent to the symbolic layer to update the belief base of the agent. On receiving the new percept (6.11), the agent merges it with the affordance (6.5) using equation (6.3). We note that the value r_1 should be bigger than r_0 which suggests the agent select the affordance (6.11) rather than the initiated affordances (6.5). Meanwhile, r_1 should be smaller than τ because (6.11) is not verified yet. By default, we instantiate $r_1 = 0.3$ in our experiments. The following rule

if `goal(des(Obj,Eff)),bel(obs(Obj),afford(Obj,Act,Eff,_,_,Rel),` $\texttt{Rel} \geq r_1, \texttt{Rel} < \tau$`))` **then** `learn(Eff,Obj,mode)`. (6.12)

indicates that the affordance learning should continue by performing `Act` when the reliability is higher than r_1 and lower than τ. The reliability of the affordance is updated by equation (6.3). The action parameter θ_{max} is selected as follows:

$$\theta_{max} = \arg\max_{\theta \in \Theta} \hat{H}(\theta) \tag{6.13}$$

which is expected to generate the maximal effect.

If the condition (6.10) cannot be satisfied, it is predicted that `Act` is not effective for achieving `Eff`. Therefore, the following message

$$\texttt{afford(Obj,Act,Eff,}0, 0, r_2\texttt{)} \tag{6.14}$$

is sent to the symbolic layer, where $0 < r_2 < r_0$. By default, we instantiate $r_2 = 0.05$. It means this affordance has a relatively low reliability for action selection. The affordances (6.5) and (6.14) are also merged using equation (6.3).

Learning when Reliability Decreases

Affordance learning can also be initiated in case that the reliability of an available affordance decreases below the threshold τ. In this case, the affordance being used for action selection is proved to be ineffective. This is likely to happen when the environment changes and the affordances change accordingly.

Termination of Affordance Learning

The learning can be terminated under certain conditions. Typically, the learning is terminated when the learned affordance is reliable enough to be used for action selection. In order to avoid endless learning, the learning will be terminated anyway if the reliability of all affordances is smaller than the threshold r_2.

6.4 Experiments

In this chapter, we investigate the integration of affordance learning and symbolic reasoning. We design a real world task to test the information flow between the symbolic layer and sub-symbolic layer in the cognitive robot control architecture (see Figure 6.1). The GOAL agent is expected to perform affordance-aware action selection, and update the statistics of the affordance knowledge through embodied robot interaction with the objects. In addition, the agent should be able to switch on/off the affordance learning process, and achieve task goals as soon as possible.

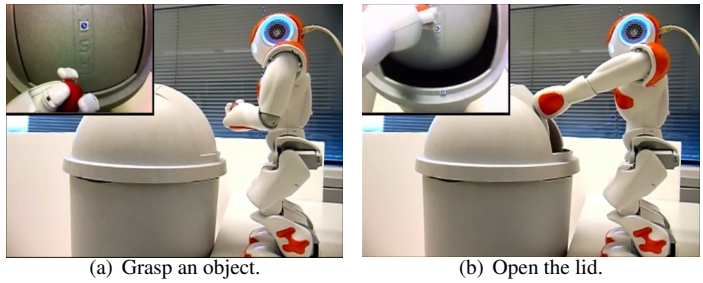
6.4.1 Task Settings

A humanoid robot NAO is used to perform a garbage disposal task (see Figure 6.2 and Figure 6.3). Two types of garbage cans are used in two experiments. In both experiments, the robot has to solve the same sequence of sub-tasks: grasp and object, open the lid, drop the object, and close the lid. The sub-task of lid opening may require affordance learning, which is decided by the robot itself.

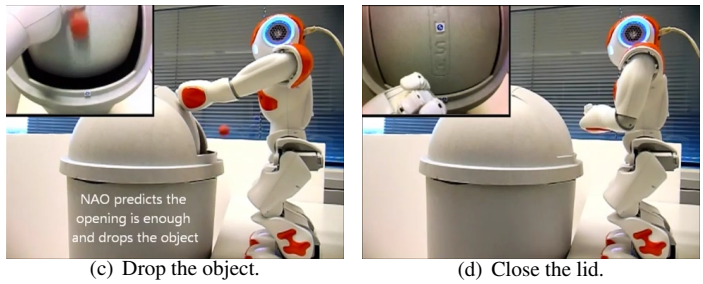
Actions

For this task, the following parameterized actions are defined for the agent :

- *grasp object*: close the hand to grasp a given object.
- *drop object*: open the hand to drop the grasped object.
- *extend arm*: move the arm forward.
- *contract arm*: move the arm backward.
- *sweep left*: move the arm left.
- *sweep right*: move the arm right.
- *lift up*: move the arm upwards.

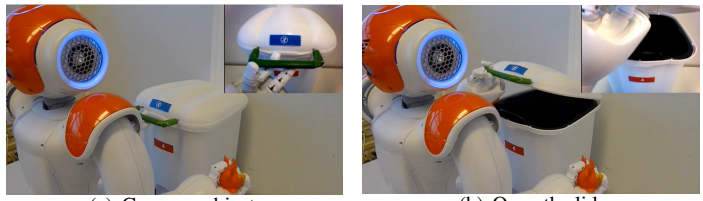


(a) Grasp an object. (b) Open the lid.

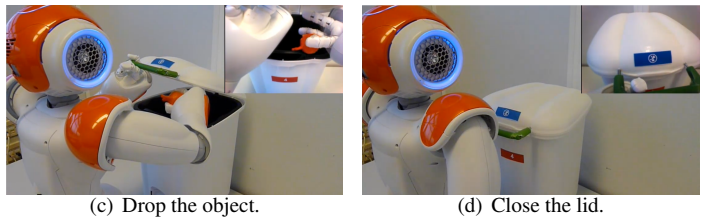


(c) Drop the object. (d) Close the lid.

Figure 6.2: Experiment 1: A humanoid robot NAO solves a garbage disposal task by pushing open the lid (two camera views). In this task, the NAO grasps an object (a red toy ball), opens the lid, drops the object, and closes the lid. It is assumed that the object is graspable and the garbage can is openable.



(a) Grasp an object. (b) Open the lid.



(c) Drop the object. (d) Close the lid.

Figure 6.3: Experiment 2: A humanoid robot NAO solves a garbage disposal task by lifting up the handle (two camera views). In this task, the NAO grasps an object (a red toy ball), opens the lid, drops the object, and contracts its arm to close the lid. It is assumed that the object is graspable and the garbage can is openable.

- *sweep down*: move the arm downwards.
- *learn*: acquire the relation between a lid/handle, an action and the `opened` effect.

The actions of arm movement are associated with low-level control programs in the environment interface (see section 6.3.1). For example, the action *extend arm* is associated with the action command "`changePositionArm(`θ`,0,0)`", where $\theta \in \Theta$ and Θ is the corresponding parameter space of the action *extend arm*. In our case, $\Theta = \{\theta \mid 0 < \theta \leq \theta_0\}$, and θ_0 is the maximal action parameter. The action command can be executed in the sub-symbolic layer to move the robot arm in the 3D Cartesian space (see Figure 3.2 for the NAO frame definition in the Cartesian space). The learning is carried out through `random` exploration in the continuous parameter space Θ. The same with chapter 5, the following action set $\mathcal{A}$={*sweep left*, *sweep right*, *push forward*, *pull back*, *lift up*, *sweep down*} is used for affordance learning.

Goals

In the task, the agent is programmed to achieve a sequence of goals. Among them, we focus on the goal of lid opening. This goal is described by the effect of `opened`, which is defined by the opened width of the lid (bigger than a given threshold, see equation (6.9)). In order to achieve this goal, the agent is expected to decide by itself whether to learn or to use the affordance knowledge in order to open the lid.

Beliefs

The agent's beliefs of the `lid` and the `opened` effect are derived from the available sensory functions in the sub-symbolic layer (NAO markers for lid recognition and color segmentation for lid openness recognition).

When the lid or handle is observed by the NAO's camera, a message "`percept(lid)`" or "`percept(handle)`" is sent to the symbolic layer. Then, "`obs(lid)`" or "`obs(handle)`" is inserted into the agent's belief base. As a result, the mental state condition `bel(obs(lid))` or `bel(obs(handle))` becomes true (see Appendix B).

The opened width of the lid is measured by equation (5.14). An effect $e \in \mathcal{E} \subseteq \mathbb{R}$ is calculated by subtracting the opened width after and before an arm movement (see equation (5.3)). For this task, we set $\delta = 0.1$ m in equation (6.9). In other words, `bel(opened)` would become true if the opened width was more than 0.1 m. Refer to Table 2.4 for how discrete effects can be obtained.

Affordance Knowledge

In the task, affordance knowledge encodes the relation between a lid/handle, an action, and the opened effect. If the conditions of rule (6.4) are satisfied, then affordance learning is necessary (Experiment 1). In contrast, previously learned affordance knowledge can be directly reused. In Experiment 2, the affordance knowledge learned in Experiment 1 will be used for action selection. Relearning might be needed if the reliability of the available affordance knowledge drops below the threshold τ.

A Trial

In each trial, the NAO starts in the same initial standing posture. It raises its arm to its camera field and grasps a given object (see Figure 6.2(a) and Figure 6.3(a)). Then, the NAO observes the garbage can and confirms that the lid is closed[3]. In order to open the lid (see Figure 6.2(b) and Figure 6.3(b)), the agent searches in its belief base for relevant affordance knowledge. If there was no affordance knowledge about how to open the lid, the agent initiates affordance learning. Otherwise, the agent selects an action to try to open the lid according to the available affordance knowledge. In this case, the selected action is sent to the sub-symbolic layer for execution, and the statistics of the affordance knowledge is updated. If the lid was opened, the NAO would drop the object (see Figure 6.2(c) and Figure 6.3(c)). Otherwise, the NAO would continue to try to open the lid. Affordance learning would be necessary if certain conditions were satisfied. Finally, the NAO closes the lid (see Figure 6.2(d) and Figure 6.3(d)). After finishing the trial, the NAO starts all over again.

6.4.2 Results

In each experiment, the NAO performs the task repeatedly in five trials, with the beliefs of affordance knowledge being updated. The robot continues a trial until it achieves the goal. In the first trial of the experiment, the robot may need to learn affordances. Later on, the robot simply uses the learned affordances to select actions, unless learning is necessary.

Experiment 1: Learning From Scratch

In this experiment, the agent initially created a set of affordances in its belief base for the lid and all actions in the action set $\mathcal{A}$. An example of affordance is `afford(lid,push forward,opened,`$0, 0, 0.1$`)` (see (6.5)).

In the first trial, the robot grasped the given object. In order to open the lid, the agent decided to initiate affordance learning according to rule (6.4). The agent selected the action *sweep left* from $\mathcal{A}$. Then, random exploration was carried out in the continuous parameter space of the action *sweep left*, i.e., in the Cartesian space $(0, \theta, 0), 0 < \theta \leq 0.1\,\text{m}$. The NAO moved the arm left with random distances for several times. Then, the following regression function was learned: $\hat{H}(\theta) = 0.006 + 0.053\,\theta$. According to the condition (6.10), the predicted maximal value $e_{max} = 0.011\,\text{m} < \delta = 0.1\,\text{m}$. In other words, the learned model predicted that the lid could not be opened by the action *sweep left*. According to equation (6.14), a message `afford(lid,sweep left,opened,`$0, 0, 0.05$`)` was sent to the agent. According to equation (6.3), the updated reliability of the affordance was 0.05. As a result, the action *sweep left* had a lower chance to be selected than the other actions.

The agent continued the learning and the action *contract arm* was selected. The action parameter space was $(\theta, 0, 0)$, $-0.1\,\text{m} \leq \theta < 0$. The learned regression function was $\hat{H}(\theta) = 0.003 - 0.021\,\theta$. According to the condition (6.10), the predicted maximal value $e_{max} = 0.005\,\text{m} < \delta = 0.1\,\text{m}$. This prediction meant that the lid could not be opened by the

[3]The lid is initially closed. The openness of the lid is recognized by color segmentation as described in the previous chapters.

action *contract arm* either. Similarly, the corresponding affordance was updated to be `afford(lid,contract arm,opened,0,0,0.05)`.

After several ineffective actions were selected, the action *extend arm* had a higher chance to be selected. The action parameter space was $(\theta, 0, 0), 0 < \theta \leq 0.1\,\text{m}$. The learned regression function was $\hat{H}(\theta) = 0.007 + 1.293\,\theta$. According to the condition (6.10), the predicted maximal value $e_{max} = 0.136\,\text{m} > \delta = 0.1\,\text{m}$. In other words, the learned model predicted that the lid could be opened by the action *extend arm*. According to equation (6.11), a message `afford(lid,extend arm,opened,0,0,0.3)` was sent to the agent. According to equation (6.3), the updated reliability of the affordance was 0.3. Currently, the conditions of rule (6.12) were satisfied. Therefore, the agent continued the learning by selecting the action *extend arm*. According to equation (6.13), the action parameter $\theta_{max} = 0.1$ was selected. After performing the action, an actual openness of $0.156\,\text{m} > \delta = 0.1\,\text{m}$ was observed. According to section 6.3.3, a message `afford(lid,extend arm,opened,1,1,1)` was sent to the agent. According to equation (6.3), the updated affordance was `afford(lid,extend arm,opened,1,1,1)`. Then, the termination condition of affordance learning was satisfied. The agent switched off the learning mode and switched on the exploitation mode. According to rule (6.2), the robot extended the arm, and the lid was opened. The object was dropped into the garbage can, and the robot contracted its arm to close the lid. The first trial ended successfully.

Since the second trial, the agent directly selected the action *extend arm* to open the lid according to rule (6.2). No learning was needed. After the 5th trial, the learned affordance became `afford(lid,extend arm,opened,5,5,1)`.

Experiment 2: Relearning When Reliability Decreases

In this experiment, two object parts `lid` and `handle` were observed. The affordance knowledge learned in Experiment 1 was reused by the agent for action selection. In order to open the lid, the agent selected the action *extend arm* upon the lid. However, this action turned out to be ineffective for lid opening. As a result, a message `afford(lid,extend arm,opened,1,0,0)` was sent to the agent. According to equation (6.3), the updated affordance was `afford(lid,extend arm,opened,6,5,0.83)`. The reliability of this affordance was still higher than the threshold $\tau = 0.6$, which meant the agent still believed *extend arm* was effective to open the lid. Therefore, the agent continued to select the action *extend arm* until the updated affordance became `afford(lid,extend arm,opened,8,5,0.57)`. The reliability 0.57 is lower than the threshold $\tau = 0.6$. At this moment, affordance learning `learn(lid,opened,random)` was initiated. However, performing all the actions in $\mathcal{A}$ upon the lid would not result in lid opening. After a few more actions upon the lid, the reliability of all affordances decreased below $r_2 = 0.05$. Therefore, the agent stopped affordance learning with the lid.

Then, the agent started the interaction with the handle. A new set of affordances was created in the belief base for the handle and all actions in the action set $\mathcal{A}$, e.g., `afford(handle,push forward,opened,0,0,0.1)` (see (6.5)). The learning continued in the similar way with Experiment 1. Finally, the affordance `afford(handle,lift up,opened,5,5,1)` was learned to open the lid by lifting up the handle.

6.5 Conclusions and Open Issues

In this chapter, a generic cognitive control architecture has been proposed for robot learning with the use of affordances. In this architecture, we have integrated affordance learning and symbolic reasoning that are carried out at different cognitive levels. Affordances are represented as symbolic knowledge maintained by a cognitive agent. The affordance knowledge can be used by the agent to select affordance-aware actions through symbolic reasoning. In addition, the symbolic affordance knowledge is updated according to a robot's own sensory-motor experience. The update of the statistics of the affordance knowledge is carried out on-line during task execution. The agent can autonomously decide when to learn or use affordances.

An experimental evaluation of the proposed architecture has been carried out. We use the agent programming language GOAL [133] to program cognitive agents in the cognitive layers, and we use Environment Interface Standard (EIS) [134] to communicate between the agents and the sub-symbolic layers. Unlike the previous chapters in which both the learning and the use of affordances take place in the sub-symbolic layers, in this chapter, the agents are responsible for using affordances to achieve goals. As a result, the agent can reason when and how to learn or use affordances in complex tasks that consists of several subtasks (e.g., garbage disposal in this chapter), rather than solving a single task (e.g., lid opening in Chapter 5). In the garbage disposal task, we have shown that the proposed architecture supports robot learning of the lid opening subtask. There is in principle no obstacle for the robot to learn other subtasks if necessary, i.e., reaching/grasping an object, dropping an object into a container, etc. However, due to the limitation of the sensory and motor functions of the NAO robot, learning all these subtasks from scratch is very challenging, especially in continuous state and action spaces. In the future, more advanced robots should be tested in more complex tasks to show the generality of the proposed architecture.

In the literature, affordances are usually learned from the sensory-motor experience in a bottom-up manner [33, 135]. We have also followed this approach throughout the earlier part of this dissertation. In contrast, the agent in this chapter maintains symbolic affordance knowledge that can be created in a top-down manner. In other words, it is possible to directly provide affordances as the additional source of knowledge to the agent, without the need of affordance learning. In such a case, assumptions are made that the affordances in the environment are well known without uncertainty, e.g., wheels are turnable and ladders are climbable [41]. However, this may not work due to environmental changes. We have shown in the second experiment that the symbolic affordance knowledge can be wrong for handling a new object. Similar to the object-action complexes (OACs) [59], we also emphasize the importance of verifying affordance knowledge before using it to achieve goals. Compared with OACs, the verification mechanism in this chapter is on-line during task execution, and the verification results will decide whether to relearn or use the affordances.

Deciding the reliability of affordance knowledge is another opportunity for future research to improve affordance-aware action selection. Currently, the initial reliability r_0, r_1, r_2 and the reliability threshold τ are assigned with default values. These values can be more adaptive to the task at hand. Consequently, the agent would make more reasonable decisions to switch between the exploration mode and the exploitation mode.

In this chapter, random exploration in continuous action parameter space has been em-

ployed, but it is possible to apply any other action exploration strategies as well. In particular, active exploration (Chapter 4) and transfer learning (Chapter 5) could be used instead.

Chapter Seven
Conclusions and Outlook

This chapter summarizes the main contributions of this dissertation and revisits the research questions posed in chapter 1. Limitations and open issues are also discussed.

7.1 Summary and Conclusions

One main objective of this dissertation is to improve the task performance of a robot through affordance learning and use. We have taken into account the efficiency of both the learning and the use of affordances to speed up task learning. Increasing the efficiency of affordance learning is important because learning may otherwise be too time consuming in practical applications. We have investigated how to speed up the learning of affordances through on-line learning (chapter 3), active learning (chapter 4), and transfer learning (chapter 5). While discussing affordance learning, we have also paid attention to the use of affordances for achieving goals. Specifically, a cognitive robot control architecture has been proposed to integrate affordance learning at the sensory-motor level and the use of affordances at the symbolic level to solve complex tasks (chapter 6). In the remainder, we summarize the main contributions of this dissertation and answer the corresponding research questions posed in chapter 1.

- **How can affordances be learned and used on-line for solving a goal-directed task?**

  Chapter 3 has introduced an approach to learn and use affordances in goal-directed tasks. We have integrated task learning, affordance learning and affordance use in a general framework, in which affordances can be obtained and used automatically during on-line task learning. Reinforcement learning (RL) has been chosen for goal-directed task learning, specifically the XCS classifier system has been used. The affordance-based XCS has shown better performance than standard XCS in real-world tasks. This is due to that the additional knowledge of affordances directly provides suggestions on action selection, without the need to wait for rewards to change the action selection policy in RL systems such as XCS. In the navigation task, we have designed a proper action filter to decrease the number of candidate actions. The experiment results have shown that using affordances is more efficient

for avoiding the selection of wrong actions that would not contribute to achieving the goal. However, it is difficult to design a generic action filter for a wide range of goal-directed tasks.

The process of data collection for affordance learning is on-line during task learning whenever an action is performed on an object. The learned affordances are also simultaneously used to speed up the task learning. Only the task-relevant affordances are learned and used, which is more efficient for solving a goal-directed task than the two-staged approach [33] that requires a robot to learn all kinds of affordances (including affordances that are irrelevant to achieving the goal) before it can use the affordances to solve the task.

We have used an affordance table to keep the latest triplets of learned affordance. It can handle dynamic environments in which the affordances may change. We have used a simple binary representation of object features without the ability to generalize over the feature space. A ternary representation used in the XCS rules can generalize learned affordances to handle new objects. However, it comes at the price of computational complexity compared to the simple affordance table.

The proposed method has been shown effective for discrete object states and robot actions, in principle there is no obstacle to handling continuous spaces. Object states and robot actions can be defined in continuous spaces, function approximators can be used to learn affordances, and a continuous version of RL algorithm can be used for task learning. However, it would be more difficult to design an action filter in continuous action spaces than in discrete action spaces.

- **How can a robot explore efficiently in continuous action spaces to learn affordances of a new object?**

  Chapter 4 has introduced active affordance learning of a new object in continuous state and action spaces. We have used function approximation to model affordances, and these models can be learned using the prediction error of action effects. Meanwhile, the prediction error also serves as a reward signal to update the action exploration policy using an actor-critic reinforcement learning structure. We have chosen feed-forward neural networks and continuous actor-critic learning automation (CACLA) to model affordances and to learn the action exploration policy, respectively. The experimental results have shown that the proposed active learning approach requires less number of exploratory actions than the random exploration baseline for the convergence of the averaged TD errors. The reason is that active exploration is carried out in a more organized way by sampling in the most uncertain areas. In contrast, random exploration occasionally samples an action resulting in high TD error which makes the averaged TD errors more difficult to converge. However, we have not considered the scalability of the active affordance learning algorithm. It will be important to further speed up the learning in higher dimensional continuous state and action spaces.

  The learned affordance models have been reused to acquire a range of manipulation skills by generating a sequence of goals in the effect space and selecting actions to achieve the goals. The goal generation continues until there is no significant effect

observed. In our experiments, a humanoid robot NAO has successfully learned how to open and close two types of garbage cans. These skills are associated with initial and goal states, therefore they are immediately reusable when other goal-directed tasks are given later on. Unlike Chapter 3 that has addressed on-line model learning and use, Chapter 4 has not considered affordance learning when using affordances to achieve goals. Nevertheless, it is still possible to enable the interaction between the learning and the use of affordances, e.g., combining active generation of goals [119] and active learning of affordances.

The proposed active affordance learning is generic and flexible that supports any perceptual proxy for representing objects, any action representation, any forward model for representing affordances, and any actor-critic RL algorithm for continuous state and action spaces. We have chosen the 2D bounding box perceptual proxy, the 3D Cartesian action space and the feed-forward neural networks, and the CACLA algorithm, respectively. However, these are not necessarily the best choices. Trying and comparing different instantiations of the modules would be interesting, e.g., using 3D point clouds to represent objects [117] and considering actions in high-dimensional spaces [119].

- **How can the learned affordances be transferred across objects to speed up the learning of a new object?**

  Chapter 5 has investigated a transfer learning approach to handle a new object with additional affordance knowledge of known objects. We have developed a part-based model to represent the affordances of complex household objects composed of several parts. Each part is associated with a set of affordance models and the corresponding set of parameterized actions. When the new object is given, the potential functional part can be found by measuring its similarity with the functional parts of known objects. An experimental evaluation has shown that transfer learning of affordances is more efficient than learning from scratch because it avoids unnecessary exploratory actions whenever possible. Specifically, transfer learning is most efficient when the new object has a more similar way of manipulation with a known object. However, transfer learning can suggest to select inappropriate actions when the new object looks similar to a known object, yet has a different way of manipulation. It is interesting to investigate how to avoid such a negative transfer.

  We have chosen linear regression models to represent affordances in the garbage can manipulation task. Non-linear models such as neural networks used in Chapter 4 would be more appropriate to describe the movement patterns of the functional parts of general household objects. However, our focus in Chapter 5 is to show the effectiveness of the transfer learning approach. There is no obstacle to use non-linear models to represent and transfer affordances. Furthermore, active learning discussed in chapter 4 could be combined with transfer learning to further speed up the exploration in the parameterized action spaces.

  We have assumed that both the initial and goal states are exactly the same, i.e., the lids are initially closed and the tasks are opening the lids wide enough. It will be necessary to consider transfer learning across objects and tasks. For instance, the experience of opening a lid could be helpful for learning how to close a door. However,

it is very challenging for the robot to acquire the concepts of a wide range of objects and tasks. The multi-task reinforcement learning (MTRL) framework [129], especially hierarchical Bayesian MTRL [130] that reuses the previously learned distribution over MDPs as an informed prior, is a promising approach to transfer learning of multiple tasks.

- **How can affordance learning be integrated with symbolic reasoning for solving complex tasks?**

  Chapter 6 has proposed a generic cognitive control architecture that integrates affordance learning and symbolic reasoning. Affordances are represented as symbolic knowledge maintained by a cognitive agent that selects affordance-aware actions through symbolic reasoning. The symbolic affordance knowledge is updated according to a robot's own sensory-motor experience during on-line task execution. We have used the agent programming language GOAL [133] to program cognitive agents which are responsible for using affordances to achieve goals. The agent can reason when and how to learn or use affordances in complex tasks that consists of several subtasks, e.g., garbage disposal. Although we have only shown that the proposed architecture supports robot learning of the lid opening subtask, there is in principle no obstacle for the robot to learn other subtasks if necessary, i.e., reaching/grasping an object, dropping an object into a container, etc. However, due to the limitation of the sensory and motor functions of the robot NAO, learning all these subtasks from scratch is very challenging, especially in continuous state and action spaces. More capable robots can be tested in more complex tasks to show the generality of the proposed architecture.

  The GOAL agent maintains symbolic affordance knowledge that can either be created in a top-down manner, or learned from the sensory-motor experience in a bottom-up manner, unlike our previous chapters or the literature that only supports the latter case [33, 135]. It means that it is possible to directly provide affordances as the additional source of knowledge to the agent, without the need of affordance learning. Nevertheless, we have also emphasized the importance of verifying the affordance knowledge before using it to achieve goals. This idea is similar with object-action complexes (OACs) [59]. However, OACs separate affordance learning and use, resulting in drawbacks that have been discussed in Chapter 3. In contrast, our verification mechanism supports on-line task execution, and the verification results can be used to decide whether to continue learning or to use the affordances.

  The cognitive control architecture supports the three learning approaches proposed in the previous chapters, i.e., on-line learning (Chapter 3), active learning (Chapter 4) and transfer learning (Chapter 5). By providing high-level action rules to the GOAL agent, affordance learning can be activated during on-line task execution whenever necessary. In addition, the high-level action rules can also specify the learning mode for the agent. For example, if the agent is learning to handle a new object, either the active learning mode or the transfer learning mode can be selected. However, it is still challenging for the agent to make such decisions without human intervention.

7.2 Open Issues and Future Research

We conclude this dissertation by reflecting on limitations of our work, and presents some interesting open issues regarding the techniques proposed in this dissertation, together with ideas of future research on learning and using affordances.

In Chapter 3, we have discussed affordance learning and use for discrete representation of states and actions, where an affordance table keeps updating the learned triplets in the form of (object, action, effect). However, this approach only maintains the latest triplet if there exist two triplets with the same pair of (object, action) but a different effect. As a result, it does not make full use of all the collected data, and it has limited support for action selection in uncertain environments. In contrast, we have considered the long-term statistics of the triplets in Chapter 6 to handle uncertainty. In other words, the reliability of an affordance is updated according to the total times and successful times of applying the affordance to select actions. Currently, we simply initialize the reliability of affordance knowledge to default values. More sophisticated mechanisms can be employed to recalculate the reliability of the affordance knowledge. For this purpose, the mechanism introduced in Chapter 3 that updates the fitness of XCS rules could be an interesting choice. Another possible choice is to use probabilistic models such as Bayesian Networks [10, 35, 44].

In Chapter 4 and 5, we have discussed active learning and transfer learning in continuous spaces, respectively. The results of chapter 4 have shown that the learning time increases with the dimension of action spaces. In our experiments, however, the state space, action space, and the effect space do not have more than three dimensions. It is necessary to consider how to scale learning in higher dimensional spaces. Besides, the models we have chosen, i.e. feed-forward neural networks and linear regression, are not guaranteed to be the most efficient ones for function approximation. Non-linear models such as Gaussian Regression is a possible choice for learning in high dimensional continuous spaces.

Chapter 4 has used forward models to represent affordances that maps from the continuous state and action spaces to the continuous effect space. Then, actions are selected using the learned forward models to generate the predicted maximal effect through an internal rehearsal mechanism. However, this involves extra computation by sampling in the action space, calculating the predicted effects, and selecting the maximal effect. It would be more convenient to use an inverse model that directly maps from the state and effect space to the action space. In our active affordance learning framework, there is no obstacle to simultaneously learn an inverse model while learning the corresponding forward model. In other words, whenever a sample of (object, action, effect) is obtained for learning the forward model by using the prediction error of effects, the same sample of (object, effect, action) can be used for learning the inverse model by using the prediction error of actions. Both of the prediction errors can be used as the reward signal for updating the actor-critic reinforcement learning module. One possible drawback of considering the inverse model is that the learning would require more training data than only learning the forward model, because the averaged TD errors are likely to converge slower.

Chapter 5 has assumed that there is only one specific goal and an object has only one functional part for the goal. More complex tasks have to be considered that there are different functional parts for different goals. We have separately modeled the movement of the part and the part-object relation. However, we have not considered the relations

between multiple parts. For example, lift/pull the handle in Figure 5.2 not only moves the handle, but also moves the lid. In the future, it is interesting to investigate the co-movement of various object parts and discover their underlying relations. Also, it will be challenging to consider the simultaneous or sequential manipulation of more than one functional parts.

The cognitive architecture proposed in Chapter 6 enables the robot to solve complex tasks that consist of several subtasks. However, the affordance learning is simply switched on and off by means of preprogrammed *if-then* action rules. An interesting research direction is to let the agent learn by itself how to control the learning to further improve the task performance. This is related to the topic of *learning to learn* [141]. The agent should be able to learn the high-level action rules. Reinforcement learning algorithms can be used to learn such rules, as has been done in Chapter 3. Nevertheless, the learning would take a long time if the robot has to learn simultaneously the symbolic level task-related concepts and action rules as well as the sensory-motor level affordances and control policies. It is necessary to focus on different aspects of the learning, following the principles of open-ended and non-task specific learning in the field of developmental robotics [1, 2, 3, 4].

This thesis has focused on fully autonomous affordance learning that does not involve human intervention once the learning is initiated. However, human-in-the-loop is required for life-long robot learning to develop human-level robotic behaviors. The supervisory control paradigm of Plan, Teach, Monitor, Intervene, and Learn (PTMIL) [50] can be considered in the future. It is an interesting topic to investigate how affordance learning can be further improved through human guidance, e.g., by observation, demonstration or tele-operation. Specifically, observing how humans use household objects can help the robot understand object affordances [13] and assist humans [46]. Demonstration is an efficient way of providing complex manipulation skills for affordance learning [49]. Tele-operation is still necessary for robots to carry out complex tasks such as turning wheels, climbing ladders and drilling a wall [41, 132]. However, the challenge of natural human-robot interaction still remains that hinders robots to learn and use affordances efficiently under human supervision. The latest progress on natural language processing (NLP) could possibly provide a common understanding for humans and robots on the concept of affordances. Previous research that associates language and affordances [15] can be extended for this purpose. In this way, robots can understand from human's perspective the meanings of objects and what can be done with them.

Bibliography

[1] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen. Autonomous mental development by robots and animals. *Science*, 291(5504):599–600, 2001.

[2] M. Lungarella, G. Metta, R. Pfeifer, and G. Sandini. Developmental robotics: a survey. *Connection Science*, 15(4):151–190, 2003.

[3] M. Asada, K. Hosoda, Y. Kuniyoshi, H. Ishiguro, T. Inui, Y. Yoshikawa, M. Ogino, and C. Yoshida. Cognitive developmental robotics: a survey. *IEEE Transactions on Autonomous Mental Development*, 1(1):12–34, 2009.

[4] A. Stoytchev. Some basic principles of developmental robotics. *IEEE Transactions on Autonomous Mental Development*, 1(2):122–130, 2009.

[5] Minoru Asada, Karl F MacDorman, Hiroshi Ishiguro, and Yasuo Kuniyoshi. Cognitive developmental robotics as a new paradigm for the design of humanoid robots. *Robotics and Autonomous Systems*, 37(2):185–193, 2001.

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

[7] W.H. Warren. Perceiving affordances: Visual guidance of stair climbing. *Journal of Experimental Psychology: Human Perception and Performance; Journal of Experimental Psychology: Human Perception and Performance*, 10(5):683, 1984.

[8] Claes von Hofsten. An action perspective on motor development. *Trends in Cognitive Sciences*, 8(6):266 – 272, 2004.

[9] 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.

[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] C. Castellini, T. Tommasi, N. Noceti, F. Odone, and B. Caputo. Using object affordances to improve object recognition. *IEEE Transactions on Autonomous Mental Development*, 3(3):207–215, 2011.

[12] T Hermans, JM Rehg, and A Bobick. Affordance prediction via learned object attributes. *International Conference on Robotics and Automation: Workshop on Semantic Perception, Mapping, and Exploration*, 2011.

[13] Hema Swetha Koppula, Rudhir Gupta, and Ashutosh Saxena. Learning human activities and object affordances from rgb-d videos. *The International Journal of Robotics Research*, 32(8):951–970, 2013.

[14] Yu Sun, Shaogang Ren, and Yun Lin. Object-object interaction affordance learning. *Robotics and Autonomous Systems*, 62(4):487–496, 2014.

[15] O. Yürüten, K. Uyanık, Y. Çalışkan, A. Bozcuoğlu, E. Şahin, and S. Kalkan. Learning adjectives and nouns from affordances on the icub humanoid robot. *From Animals to Animats 12*, pages 330–340, 2012.

[16] Marcus Oladell and Manfred Huber. Symbol generation and feature selection for reinforcement learning agents using affordances and u-trees. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 657–662. IEEE, 2012.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Marti Sanchez-Fibla, Armin Duff, and Paul F.M.J. Verschure. The acquisition of intentionally indexed and object centered affordance gradients: A biomimetic controller and mobile robotics benchmark. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1115–1121, sept. 2011.

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

[23] A.K. Bozcuoglu and E. Sahin. Traversability on a simple humanoid: What did i just trip over? In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 701 –706, oct. 2011.

[24] C. de Granville, J. Southerland, and A.H. Fagg. Learning grasp affordances through human demonstration. In *Proceedings of the International Conference on Development and Learning (ICDL)*, 2006.

[25] J.D. Sweeney and R. Grupen. A model of shared grasp affordances from demonstration. In *7th IEEE-RAS International Conference on Humanoid Robots*, pages 27–35. IEEE, 2007.

[26] R. Detry, E. Baseski, M. Popovic, Y. Touati, N. Kruger, O. Kroemer, J. Peters, and J. Piater. Learning object-specific grasp affordance densities. In *8th International Conference on Development and Learning*, pages 1–7. IEEE, 2009.

[27] C. Barck-Holst, M. Ralph, F. Holmar, and D. Kragic. Learning grasping affordance using probabilistic and ontological approaches. In *International Conference on Advanced Robotics (ICAR)*, pages 1 –6, june 2009.

[28] Emre Ugur, Erhan Oztop, and Erol Sahin. Going beyond the perception of affordances: Learning how to actualize them through behavioral parameters. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4768–4773. IEEE, 2011.

[29] E. Ugur, E. Sahin, and E. Oztop. Self-discovery of motor primitives and learning grasp affordances. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 3260–3267. IEEE, 2012.

[30] Tao Geng, James Wilson, Michael Sheldon, Mark Lee, and Martin Hülse. Synergy-based affordance learning for robotic grasping. *Robotics and Autonomous Systems*, 61(12):1626–1640, 2013.

[31] M. Lopes, F.S. Melo, and L. Montesano. Affordance-based imitation learning in robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1015–1021. IEEE, 2007.

[32] 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.

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

[34] Verica Krunic, Giampiero Salvi, Alexandre Bernardino, Luis Montesano, and José Santos-Victor. Affordance based word-to-meaning association. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4138–4143. IEEE, 2009.

[35] B. Moldovan, P. Moreno, M. van Otterlo, J. Santos-Victor, and L. De Raedt. Learning relational affordance models for robots in multi-object manipulation tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4373–4378. IEEE, 2012.

[36] A. Stoytchev. Behavior-grounded representation of tool affordances. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 3060 – 3065, April 2005.

[37] V Tikhanoff, U Pattacini, L Natale, and G Metta. Exploring affordances and tool use on the icub. In *13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 130–137. IEEE, 2013.

[38] Tucker Hermans, James M Rehg, and Aaron F Bobick. Decoupling behavior, perception, and control for autonomous learning of affordances. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4989–4996. IEEE, 2013.

[39] Maurice Fallon, Scott Kuindersma, Sisir Karumanchi, Matthew Antone, Toby Schneider, Hongkai Dai, Claudia Pérez D'Arpino, Robin Deits, Matt DiCicco, Dehann Fourie, et al. An architecture for online affordance-based perception and whole-body planning. *Journal of Field Robotics*, 32(2):229–254, 2015.

[40] Stephen Hart, Paul Dinh, and Kim Hambuchen. Affordance templates for shared robot control. In *AAAI Fall Symposium on AI for Human-Robot Interaction*, 2014.

[41] Stephen Hart, Paul Dinh, and Kimberly Hambuchen. The affordance template ros package for robot task programming. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 6227–6234. IEEE, 2015.

[42] Emre Ugur, Sandor Szedmak, and Justus Piater. Bootstrapping paired-object affordance learning with learned single-affordance features. In *2014 joint IEEE International Conferences on Development and Learning and Epigenetic Robotics (ICDL-Epirob)*, pages 476–481. IEEE, 2014.

[43] 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.

[44] Afonso Gonçalves, João Abrantes, Giovanni Saponaro, Lorenzo Jamone, and Alexandre Bernardino. Learning intermediate object affordances: Towards the development of a tool concept. In *2014 Joint IEEE International Conferences on Development and Learning and Epigenetic Robotics (ICDL-Epirob)*, pages 482–488. IEEE, 2014.

[45] Tucker Hermans, Fuxin Li, James M Rehg, and Aaron F Bobick. Learning stable pushing locations. In *2013 IEEE Third Joint International Conference on Development and Learning and Epigenetic Robotics (ICDL-epirob)*, pages 1–7. IEEE, 2013.

[46] Hema Swetha Koppula and Ashutosh Saxena. Anticipating human activities using object affordances for reactive robotic response. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):14–29, 2016.

[47] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. *Handbook of robotics*, 1, 2008.

[48] Stefan Schaal, Jan Peters, Jun Nakanishi, and Auke Ijspeert. Learning movement primitives. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, pages 561–572. Springer Berlin/Heidelberg, 2005.

[49] Sang Hyoung Lee and Il Hong Suh. Skill learning and inference framework. In *Artificial General Intelligence*, volume 7999 of *Lecture Notes in Computer Science*, pages 196–205. Springer Berlin Heidelberg, 2013.

[50] Thomas B Sheridan. *Telerobotics, automation, and human supervisory control*. MIT press, 1992.

[51] Iman Awaad, Gerhard K Kraetzschmar, and Joachim Hertzberg. Socializing robots: The role of functional affordances. In *International Workshop on Developmental Social Robotics (DevSoR): Reasoning about Human, Perspective, Affordances and Effort for Socially Situated Robots at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[52] S. Griffith, J. Sinapov, M. Miller, and A. Stoytchev. Toward interactive learning of object categories by a robot: A case study with container and non-container objects. In *Proceedings of the 8th International Conference on Development and Learning*, pages 1–6, june 2009.

[53] Emre Ugur and Justus Piater. Emergent structuring of interdependent affordance learning tasks. In *2014 Joint IEEE International Conferences on Development and Learning and Epigenetic Robotics (ICDL-Epirob)*, pages 489–494. IEEE, 2014.

[54] Tucker Hermans. *Representing and Learning Affordance-Based Behaviors*. PhD thesis, Georgia Institute of Technology, 2014.

[55] Chang Wang, Koen V. Hindriks, and Robert Babuska. Robot learning and use of affordances in goal-directed tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2288–2294. IEEE, 2013.

[56] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.

[57] Chang Wang, Koen Hindriks, and Robert Babuska. Active learning of affordances for robot use of household objects. In *14th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 566–572. IEEE, Nov 2014.

[58] C. Wang, K.V. Hindriks, and R. Babuska. Effective transfer learning of affordances for household robots. In *2014 Joint IEEE International Conferences on Development and Learning and Epigenetic Robotics (ICDL-Epirob)*, pages 469–475. IEEE, Oct 2014.

[59] Norbert Krüger, Christopher Geib, Justus Piater, Ronald Petrick, Mark Steedman, Florentin Wörgötter, Aleš Ude, Tamim Asfour, Dirk Kraft, Damir Omrčen, et al. Object–action complexes: Grounded abstractions of sensory–motor processes. *Robotics and Autonomous Systems*, 59(10):740–757, 2011.

[60] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2012.

[61] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[62] Vladimir Vapnik, Steven E Golowich, and Alex Smola. Support vector method for function approximation, regression estimation, and signal processing. *Advances in neural information processing systems*, pages 281–287, 1997.

[63] Dan W Patterson. *Artificial neural networks: theory and applications*. Prentice Hall PTR, 1998.

[64] George AF Seber and Alan J Lee. *Linear regression analysis*, volume 936. John Wiley & Sons, 2012.

[65] Carl Edward Rasmussen. Gaussian processes for machine learning. MIT Press, 2006.

[66] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

[67] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[68] Changyun Wei, Junchao Xu, Chang Wang, Pascal Wiggers, and Koen V. Hindriks. An approach to navigation for the humanoid robot nao in domestic environments. In *14th Towards Autonomous Robotic Systems (TAROS)*, pages 298–310, 2013.

[69] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

[70] Pierre-Yves Oudeyer, Frederic Kaplan, et al. How can we define intrinsic motivation? In *proceedings of the 8th international conference on epigenetic robotics: modeling cognitive development in robotic systems*, 2008.

[71] Andrew G Barto, Satinder Singh, and Nuttapong Chentanez. Intrinsically motivated learning of hierarchical collections of skills. In *Proceedings of the 3rd International Conference on Development and Learning (ICDL)*, pages 112–19, 2004.

[72] MT Turvey. Affordances and prospective control: An outline of the ontology. *Ecological psychology*, 4(3):173–187, 1992.

[73] M. Steedman. Plans, affordances, and combinatory grammar. *Linguistics and Philosophy*, 25(5):723–753, 2002.

[74] T.A. Stoffregen. Affordances as properties of the animal-environment system. *Ecological Psychology*, 15(2):115–134, 2003.

[75] A. Chemero. An outline of a theory of affordances. *Ecological Psychology*, 15(2):181–195, 2003.

[76] T.E. Horton, A. Chakraborty, and R.S. Amant. Affordances for robots: a brief survey. *The Journal of the Philosophical-Interdisciplinary Vanguard*, pages 70–84, 2012.

[77] K.S. Jones. What is an affordance? *Ecological Psychology*, 15(2):107–114, 2003.

[78] Serge Thill, Daniele Caligiore, Anna M Borghi, Tom Ziemke, and Gianluca Baldassarre. Theories and computational models of affordance and mirror systems: an integrative review. *Neuroscience & BioBehavioral Reviews*, 37(3):491–521, 2013.

[79] R.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.

[80] E. Erdemir, C.B. Frankel, K. Kawamura, S.M. Gordon, S. Thornton, and B. Ulutas. Towards a cognitive robot that uses internal rehearsal to learn affordance relations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2016–2021. IEEE, 2008.

[81] J. Sun, J.L. Moore, A. Bobick, and J.M. Rehg. Learning visual object categories for robot affordance prediction. *The International Journal of Robotics Research*, 29(2-3):174–197, 2010.

[82] Dan Pelleg, Andrew W Moore, et al. X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML*, pages 727–734, 2000.

[83] B. Akgun, N. Dag, T. Bilal, I. Atil, and E. Sahin. Unsupervised learning of affordance relations on a humanoid robot. In *24th International Symposium on Computer and Information Sciences(ISCIS)*, pages 254 –259, sept. 2009.

[84] J. Sinapov and A. Stoytchev. Detecting the functional similarities between tools using a hierarchical representation of outcomes. In *7th IEEE International Conference on Development and Learning (ICDL)*, pages 91–96, aug. 2008.

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

[86] Jianbo Shi and Carlo Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[87] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006*, pages 404–417. Springer, 2006.

[88] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[89] S. Ivaldi, Sao Mai Nguyen, N. Lyubova, A. Droniou, V. Padois, D. Filliat, P.-Y. Oudeyer, and O. Sigaud. Object learning through active exploration. *IEEE Transactions on Autonomous Mental Development*, 6(1):56–72, March 2014.

[90] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

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

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

[93] 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.

[94] Pier Luca Lanzi. Learning classifier systems: then and now. *Evolutionary Intelligence*, 1:63–82, 2008.

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

[96] Pier Luca Lanzi and Daniele Loiacono. Xcslib: The xcs classifier system library, 2009.

[97] B.D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[98] Stefan Schaal, Peyman Mohajerian, and Auke Ijspeert. Dynamics systems vs. optimal control–a unifying view. *Progress in brain research*, 165:425–445, 2007.

[99] Jens Kober and Jan R Peters. Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, pages 849–856, 2009.

[100] Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. Robot motor skill coordination with em-based reinforcement learning. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3232–3237. IEEE, 2010.

[101] Jens Kober, Andreas Wilhelm, Erhan Oztop, and Jan Peters. Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots*, 33(4):361–379, 2012.

[102] Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3):263–279, 2013.

[103] Richard M Ryan and Edward L Deci. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary educational psychology*, 25(1):54–67, 2000.

[104] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[105] Johannes Kulick, Marc Toussaint, Tobias Lang, and Manuel Lopes. Active learning for teaching a robot grounded relational symbols. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1451–1457. AAAI Press, 2013.

[106] Maya Cakmak, Crystal Chao, and Andrea L Thomaz. Designing interactions for robot active learners. *IEEE Transactions on Autonomous Mental Development*, 2(2):108–118, 2010.

[107] Shiwali Mohan and John E. Laird. Learning goal-oriented hierarchical tasks from situated interactive instruction. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 387–394, 2014.

[108] Andrew G Barto. Intrinsic motivation and reinforcement learning. In *Intrinsically Motivated Learning in Natural and Artificial Systems*, pages 17–47. Springer, 2013.

[109] Gianluca Baldassarre and Marco Mirolli. Intrinsically motivated learning systems: an overview. In *Intrinsically Motivated Learning in Natural and Artificial Systems*, pages 1–14. Springer, 2013.

[110] Goren Gordon, Ehud Fonio, and Ehud Ahissar. Learning and control of exploration primitives. *Journal of computational neuroscience*, pages 1–22, 2014.

[111] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *arXiv preprint cs/9603104*, 1996.

[112] Adrien Baranès and P-Y Oudeyer. R-iac: Robust intrinsically motivated exploration and active learning. *IEEE Transactions on Autonomous Mental Development*, 1(3):155–169, 2009.

[113] P-Y Oudeyer, Frédéric Kaplan, and Verena Vanessa Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286, 2007.

[114] Stephen Hart and Roderic Grupen. Intrinsically motivated affordance discovery and modeling. In *Intrinsically Motivated Learning in Natural and Artificial Systems*, pages 279–300. Springer, 2013.

[115] Michael I Jordan and David E Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive science*, 16(3):307–354, 1992.

[116] Daniel M Wolpert and Mitsuo Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11(7):1317–1329, 1998.

[117] Simon Christoph Stein, Florentin Wörgötter, Markus Schoeler, Jeremie Papon, and Tomas Kulvicius. Convexity based object partitioning for robot applications. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3213–3220. IEEE, 2014.

[118] Hado van Hasselt and Marco A Wiering. Using continuous action spaces to solve discrete problems. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1149–1156. IEEE, 2009.

[119] Adrien Baranes and Pierre-Yves Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73, 2013.

[120] Lisa Torrey and Jude Shavlik. Transfer learning. *Handbook of Research on Machine Learning Applications. IGI Global*, 3:17–35, 2009.

[121] S. Thrun and T. Mitchell. Learning one more thing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI)*, San Mateo, CA, 1995. Morgan Kaufmann.

[122] Rajat Raina, Andrew Y. Ng, and Daphne Koller. Constructing informative priors using transfer learning. In *In Proceedings of the 23rd International Conference on Machine Learning*, pages 713–720, 2006.

[123] Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. Transferring naive bayes classifiers for text classification. In *Proceedings of the national conference on artificial intelligence*, volume 22, pages 540–545. AAAI, 2007.

[124] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

[125] Matthew E Taylor, Peter Stone, and Yaxin Liu. Value functions for rl-based behavior transfer: A comparative study. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, pages 880–885. AAAI, 2005.

[126] Lisa Torrey, Jude Shavlik, Sriraam Natarajan, Pavan Kuppili, and Trevor Walker. Transfer in reinforcement learning via markov logic networks. In *AAAI Workshop on Transfer Learning for Complex Tasks*, 2008.

[127] A Goncalves, G. Saponaro, L. Jamone, and A Bernardino. Learning visual affordances of objects and tools through autonomous robot exploration. In *2014 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 128–133, May 2014.

[128] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *The Journal of Machine Learning Research*, 6:1939–1959, 2005.

[129] S. P. Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8:323–339, 1993.

[130] Soumya Ray Prasad Tadepalli Aaron Wilson, Alan Fern. Multi-task reinforcement learning:a hierarchical bayesian approach. In *Proceedings of the 24 th International Conference on Machine Learning*, pages 1015–1022. ACM, 2007.

[131] Iman Awaad, Gerhard K Kraetzschmar, and Joachim Hertzberg. Affordance-based reasoning in robot task planning. In *Planning and Robotics (PlanRob) Workshop ICAPS-2013*, 2013.

[132] Gabriel Barth-Maron, David Abel, James MacGlashan, and Stefanie Tellex. Affordances as transferable knowledge for planning agents. In *2014 AAAI Fall Symposium Series*, 2014.

[133] Koen V Hindriks. Programming rational agents in goal. In *Multi-Agent Programming:*, pages 119–157. Springer, 2009.

[134] Tristan M Behrens, Koen V Hindriks, and Jürgen Dix. Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, 61(4):261–295, 2011.

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

[136] John E Laird, Allen Newell, and Paul S Rosenbloom. Soar: An architecture for general intelligence. *Artificial intelligence*, 33(1):1–64, 1987.

[137] John Laird. *The Soar cognitive architecture*. MIT Press, 2012.

[138] Michael Beetz, Lorenz Mosenlechner, and Moritz Tenorth. Cram–a cognitive robot abstract machine for everyday manipulation in human environments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1012–1017. IEEE, 2010.

[139] Anand S Rao, Michael P Georgeff, et al. Bdi agents: From theory to practice. In *Proceedings of the First International Conference on Multiagent Systems (ICMAS)*. AAAI, 1995.

[140] Leon Sterling. *The art of Prolog: advanced programming techniques*. MIT press, 1994.

[141] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

Appendix A
Extended Classifier System (XCS)

Parts of this section have been published in [95]. An overview of XCS used on NAO is given in Figure 7.1.

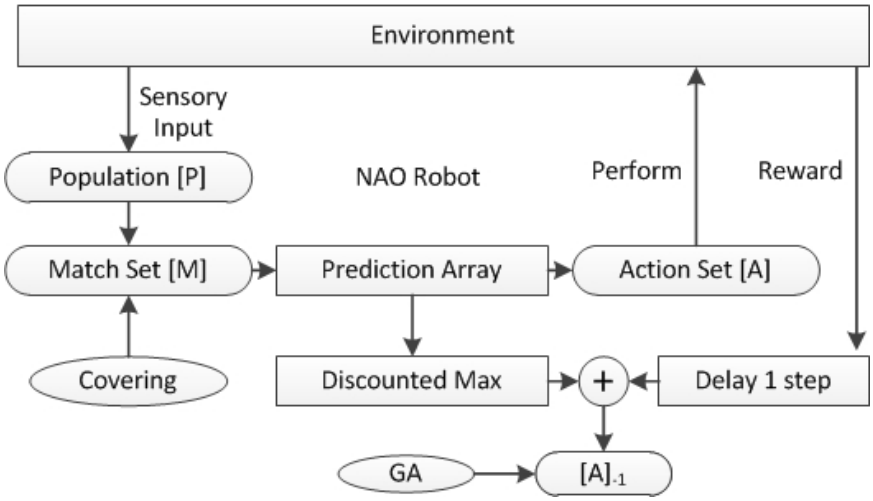


Figure 7.1: An overview of XCS

Performance Component

The performance component controls the system's behavior. In Wilson's XCS, either a pure explore trial or a pure exploit trial is performed on each time step. Usually, the system alternates between the two. In other words, learning happens only in explore mode while evaluation happens in exploit mode.

Explore Trials

A random action is usually selected from the rules in the match set [M] to match the current sensory input. However, random actions are inefficient for robots, e.g., moving back and forwards without changing its current state. Therefore, consistent actions are adopted to avoid learning in situations where the state of the robot remains the same after one action. In other words, LCS matches an input to create a match set [M] and an action set [A], then performs the selected action until there is a significant change in sensory input. If the new input still matches all the classifiers in [A], then it continues with the current action. Else if none of the classifiers in [A] matches the current input, [A] is deleted. Otherwise, a probabilistic selection takes place among them.

Exploit Trials

The system deterministically selects the highly recommended action with the highest prediction so that an optimal action sequence is chosen to achieve the goal state. However, if the environment changes, the system needs to relearn the optimal policy. Traditionally, when some accurate classifiers suddenly become inaccurate, the system decides that the environment has changed. But this change might be not significant enough that the old optimal policy still applies to achieve the goal state. A more natural way is to carry out the old optimal action sequence after the environmental change and check if the goal state is still achieved. This requires a change of the explore/exploit framework such that an extra exploit trial is needed after the goal state is achieved in one exploit trial. Otherwise, the system needs to keep a record of every action sequence and store the optimal one, which is also possible but involves additional memory operations. As XCS has no internal memory, external memory is needed to store the goal state and the number of action steps towards it.

Reinforcement Component

The reinforcement (or credit assignment) component distributes the incoming reward among the classifiers. For sequential tasks, updates only occur in the previous time step's action set $[A]_{-1}$ because they make use of the prediction array on the following time step (see equation (7.2)), except that during the last trial of an episode, both [A] and $[A]_{-1}$ are updated. General *Widrow-Hoff learning rule* is used:

$$p_j(a) \leftarrow p_j(a) + \beta_p(P - p_j(a)) \tag{7.1}$$

where $p_j(a)$ is the system prediction of classifier j in $[A]_{-1}$ if its action a is performed, and $0 < \beta_p \leq 1$ is the learning rate controlling the prediction updates. P is the weighted sum of the previous time step's reward r_{t-1} and the maximal system prediction $P(a_i)$ for action a_i:

$$P = r_{t-1} + \gamma \max_i P(a_i) \tag{7.2}$$

where γ is the discount rate [92].

Rule Discovery Component

A rule discovery component applies a Genetic Algorithm (GA) to the classifiers to update the current knowledge. In sequential tasks, GA invocations occur in $[A]_{-1}$ where the

Niche GA is triggered, which is a restricted mating scheme only happening among related classifiers. This process is described as :

$$\frac{\Sigma_{x\in[A]}(t - GA_x) \times numerosity(x)}{\Sigma_{x\in[A]}\, numerosity(x)} > \theta_{GA} \tag{7.3}$$

where t is the current time step and GA_x is the time step when the classifier x was created but never been in such an $[A]$ or was last in an action set in which the GA was invoked. The *numerosity* of x indicates the number of classifiers with the same conditions and actions. θ_{GA} is a threshold.

When Niche GA is triggered, two parent classifiers are selected with a probability proportional to their fitnesses, usually *Roulette Wheel Selection* is used in practice [85]. Standard crossover and mutation operators perform on them to get two offspring classifiers. Then, GA subsumption takes over to check if the condition part is logically subsumed by the condition of an accurate and sufficiently experienced parent.

The fitness calculation is the main difference between accuracy-based XCS and traditional strength-based LCS. When a classifier j is created, its initial prediction is denoted as p_j^0, prediction error as ϵ_j^0 and fitness as F_j^0. Then, p_j is updated by equation (7.1), ϵ_j updated by equation (7.4), and F_j updated by equation (7.5).

$$\epsilon_j \leftarrow \epsilon_j + \beta_e(|P - p_j| - \epsilon_j) \tag{7.4}$$

where ϵ_j is the prediction error of classifier j, β_e is the error learning rate, $|P - p_j|$ is the target prediction error towards which ϵ_j is updated.

$$F_j \leftarrow F_j + \beta_f(\kappa_j^{'} - F_j) \tag{7.5}$$

where β_f is the fitness learning rate, classifier j's fitness F_j is updated towards its relative accuracy $\kappa_j^{'}$ calculated by equation (7.6) and (7.7):

$$\kappa_j = \begin{cases} 1 & \text{if} \quad \epsilon_j < \epsilon_0 \\ \alpha(\epsilon_j/\epsilon_0)^{-v} & \text{otherwise} \end{cases} \tag{7.6}$$

where ϵ_0 is a threshold that decides all classifiers' accuracies are equal if their prediction errors are below it, or decreased by α and v. Once $\kappa_j, j \in [A]$ have been updated, each classifier's relative accuracy $\kappa_j^{'}$ is updated by:

$$\kappa_j^{'} = \frac{\kappa_j \times numerosity(j)}{\Sigma_{x\in[A]}\, \kappa_x \times numerosity(x)} \tag{7.7}$$

which normalizes the accuracies so that they sum up to 1. In short, a classifier's fitness is an inverse function of its prediction error, which is ignored if below ϵ_0.

Parameter Settings

The parameter settings take a reference of Wilson's *woods2* example [85] (see Table 7.1). The main difference is the separate learning rates that β_p is for prediction update, β_ϵ for error update and β_f for fitness update. All of them are initialized with the default learning

Table 7.1: XCS parameter settings for NAO Searching Task

Parameter	Notation	Value
Population size	N	100
discount factor	γ	0.7
Prediction learning rate	β_p	0.2
Error learning rate	β_ϵ	0.2
Fitness learning rate	β_f	0.2
Crossover probability	χ	0.8
Mutation probability	μ	0.04
Accuracy criterion	ϵ_0	10
Accuracy falloff rate	α	0.1
Accuracy exponent	v	5
Prediction initial	p^0	10
Error initial	ϵ^0	0
Fitness initial	f^0	0.01
Trial step limit	N_s	20
Hash probability	$P_\#$	0.6
GA threshold	θ_{GA}	5
Deletion threshold	θ_{Del}	20

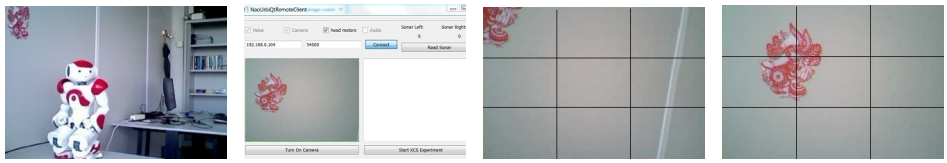
rate 0.2 and can be changed in dynamic experiments. Other changes of the parameters include the population size limit and GA threshold, both are decreased due to much less experiment trials compared with the thousands of trials in *woods2*. In our experiments, the movement boundary is $[-0.5, 0.5]\times[-0.5, 0.5]$, in other words, it is a 11×11 maze in which the NAO always starts from $(0, 0)$. In case of endless search, the maximal step in one trial is no more than $N_s = 20$.

Object searching and tracking

In the second experiment, NAO was placed on a table and the task was to find and track a target object in the camera view (see Figure 7.2). At first, NAO moved its head to search for the object. Then, the object location would be changed after NAO had confirmed to find it. NAO was expected to learn an action selection policy to quickly find the target object in such a dynamic environment.

As this sequential decision problem involved multi-step actions towards a task goal, reinforcement learning (RL) provided a good framework to solve such a problem. The state, action and reward were defined as follows:

- A state was represented as a 9-bit binary string. Each bit of the string was associated with a segmented part of an input image (see Figure 7.2(c) - 7.2(d)). A bit took value



(a) NAO on a table (b) GUI (c) Part of target found (d) One goal state

Figure 7.2: Object searching and tracking task environment.

1 if there was a color blob in the corresponding image part; otherwise, it took value 0. For example, the strings of Figure 7.2(c) and Figure 7.2(d) were '100000000' and '110110000', respectively.

- An action was encoded as a 2-bit binary string. Each bit corresponded to a head motor. The head pitched or yawed within the range of [-0.67, 0.51] or [-2.079, 2.079] (in radials). The two motors were initiated to the resting position (0 radial). On each head movement, 0.1 radial was changed on one motor. As a result, the four actions were '00' (look up), '01' (look right), '10' (look down) and '11' (look left). Whenever the motor limits were reached, the head was reset to the initial state.

- A reward of 1000 was given when the object was observed at the image center, i.e., the fifth bit of the state string was 1. Otherwise, no reward was given.

Specifically, an Extended Classifier System (XCS) [85] was implemented[1] for policy learning in this RL task. XCS is a rule-based system. It can be regarded as a generalization of tabular Q-learning by using a Genetic Algorithm (GA) [91] to aggregate equivalent states in the Q-table [92]. Besides, for physical robot control, XCS does not require careful tuning of parameters to achieve satisfactory behavior [93]. Therefore, XCS was chosen for the current task as well as for the RL tasks in Chapter 3.

The knowledge of XCS is represented as a set of rules[2]. In this dissertation, the classical ternary representation {0,1,#} is used 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. An XCS rule maps a condition and an action to a prediction, with an associated fitness as follows:

$$(\text{condition}, \text{action}) \rightarrow \text{prediction} : \text{fitness} \tag{7.8}$$

For example, the rule $(0\#0\#11\#11, 01) \rightarrow 1000 : 0.59$ means if the current state string meets the condition 0#0#11#11 and if action 01 is taken, then a reward of 1000 is predicted. This rule has a fitness of 0.59.

In XCS, the rules are value function fragments, and XCS generalizes over its value function using GA techniques. 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

[1] http://sourceforge.net/projects/xcslib/

[2] They are also called classifiers in the LCS literature [94]

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.

An experiment consisted of several trials and NAO started a trial from the resting state. While the goal state was not achieved and the motor limits were not reached, NAO was in the exploration mode[3]. It captured an image to obtain the current state before selecting one action and observing the consequent state. This perception and action loop continued till the end of the experiment. In case of endless searching, a trial ended anyway after a certain number of action steps. Table 7.2 lists some of the learned rules.

Table 7.2: An example of XCS rules

Condition	Action	Prediction	Error	Fitness	Exp.	Num.
1001#0000	01	10	0	0.01	0	1
#0###0##0	11	355.7	95.1	0.99	13	5
1#00#0###	00	453.6	116.2	0.94	11	1
0##00#00#	01	262.6	40.1	0.41	16	9
0#0#01##0	01	524.9	139.4	0.62	14	3

At first, the object was located on the up left side of NAO. It took 4 action steps, i.e., 2 action looking left (‘11’) and 2 actions looking up (‘00’) , to achieve the task goal. The first rule was initiated and not updated yet. The second rule was the most experienced one and had the highest fitness value. It advocated action ‘11’ (look left) when the target object was out of the camera view. The third rule encouraged action ‘00’ (look up) when the target object was observed in the left up corner (the first bit of the state string was 1).

Then, the object was moved to the right side of NAO which needed 3 actions looking right to find it. In this new case, some of the learned rules gave wrong suggestions for action selection, e.g., the second rule in Table 7.2. In the beginning, NAO always looked left using this rule without finding the object. As a result, this rule became inaccurate and its fitness value dropped. Meanwhile, the fitness of the fourth rule in Table 7.2 increased and suggested turn right (‘01’). The last rule also encouraged turning right when the object was observed in the right part of the camera view. This rule had a high prediction value because this state was near the goal state.

This experiment has showed how an RL system solved a goal-directed task through online interaction between the robot and the environment. The innate generalization mechanism of XCS enables learning general and accurate rules, which are at the same time readable for humans. This provides a good framework for the reuse of learned knowledge in the long term robot development.

[3] A variety of exploration strategies were also tested in the experiments. Refer to [95] for the details.

Appendix B
Programming GOAL Agents

In the sequel, we introduce the basic features of a GOAL agent program. Refer to [133] for details about GOAL agents.

Programming GOAL agents can be seen as programming with *mental states*. The mental state of a GOAL agent consists of its *knowledge*, *beliefs* and *goals*. The *knowledge base* contains conceptual or domain knowledge which is assumed to be static and unchanged in the given task. Unlike the knowledge base, the *belief base* contains the agent's beliefs about the current state of its environment. The beliefs are dynamic and changing according to environmental changes. Another feature of GOAL is that multiple declarative goals can be maintained that are to be achieved sequentially by the agent. A *blind commitment strategy* is used by GOAL agents, which means that an agent commits to its goals and drops the goals only when they have been achieved.

A GOAL agent needs to be able to *inspect* its current mental state. For this purpose, special conditions called *mental state conditions* have to be programmed. These conditions are useful in rules to specify a strategy for action selection. A mental state condition consists of conditions on the belief base or the goal base. It can be represented as a conjunction of mental atoms of the form `bel(`ϕ`)` and `goal(`ϕ`)`, where ϕ is a conjunction of literals. For example, `bel(openness(Size))` expresses "the agent believes that the size of openness is `Size`".

In order to achieve its goals, an agent needs to choose actions to interact with its environment. These actions are programmed in a symbolic form which will be translated into low-level control programs and executed by the robot's end-effectors. For example, a push action `push(Dist)` means to push an object `Obj` forward with a distance of `Dist`. In an *action specification* section of a GOAL agent, preconditions and postconditions can be specified for each action. The *precondition* is a conjunction of literals that are used to evaluate whether it is possible to perform the *action*. Only when the precondition is believed to hold, the corresponding action is *enabled* to be selected by the agent. For example, having reached an object is the precondition of pushing it, and having grasped a handle is the precondition of pulling it. Similar to a precondition, a *postcondition* is also a conjunction of literals. It specifies the action effects that are expected to change the mental state of the agent.

By means of *action rules*, the strategy is specified for the agent to select actions. An action rule informs a GOAL agent when it is an opportunity to perform an action. Such a

rule is of the form **if** *msc* **then** *action*, where *msc* is a mental state condition, and *action* is an action. The mental state condition *msc* must be checked before the corresponding *action* can be selected. The *action* is said to be *applicable* if an instantiation of *msc* is true. In addition, the *action* can be selected only if it is also enabled, i.e., its precondition is satisfied.

After action selection, the agent needs to update its belief base for new decision making. The *event module* of an agent program is responsible for this update. Specifically, the belief base can be updated by querying the belief base as well as the database of *percepts*. This database maintains the perceptual knowledge about the environment. A rule of the form **if** `bel(percept(`ϕ`))` **then** `insert` ϕ can be used to add new beliefs to the belief base after performing the query `bel(percept(`ϕ`))`. Also, a belief ϕ can be deleted from the belief base using the command `delete(`ϕ`)`.

Summary

The learning capability is essential for service robots to develop useful manipulation skills and solve household tasks. It is useful for robots to learn object affordances, which provide information about potential action effects on objects. This information is task-independent and can be used to select actions for solving a variety of tasks. In this dissertation, we are interested in efficient robot learning and use of affordances. The learning and the use of affordances are considered together rather than separated as two independent stages. Robots have to cope with changing environments that require affordance learning whenever necessary. In addition, the efficiency should be taken into account because affordance learning through embodied robot interaction with objects is typically time-consuming in order to collect enough training data. Continuous action spaces provide infinitely many action choices which make the data collection more difficult. Moreover, it is not efficient for the robot to learn every object from scratch. The robot needs to reuse relevant past experience to improve its current performance of task execution.

This dissertation aims to improve the efficiency of affordance learning and use. We have considered three learning mechanisms that can speed up collecting data and solving tasks. First, we have proposed *on-line learning of affordances* that enables on-line data collection whenever the robot applies an effective action on the objects. Meanwhile, the learned affordances can be used to avoid undesired actions in the reinforcement learning framework in which goal-directed tasks are formulated. Second, we have proposed *active learning of affordances* that speeds up the data collection by active exploration in continuous action spaces. Affordance models are learned to predict action effects in continuous spaces while the prediction error is served as the reward signal to update the action selection policy. Third, we have proposed *transfer learning of affordances* that reuses the learned affordances of relevant objects to speed up the learning of a new task. The robot decides by itself not only whether the transfer learning should happen, but also how to adjust its action selection strategy. We have demonstrate through real-world experiments with the humanoid robot NAO that the proposed affordance learning methods are more efficient than previous approaches in the literature.

Finally, we have proposed an agent-based robot control architecture that facilitates affordance learning and reasoning at different cognitive levels. In contrast to affordance learning that takes place at the sub-symbolic level through embodied robot interaction with objects, reasoning takes place at a higher symbolic level. These two levels effectively interact that affordance learning is controlled by the cognitive layer, while the affordance knowledge saved in the cognitive layer is grounded in the robot's own sensory-motor ex-

perience. The agent can autonomously decide by itself when to switch on/off affordance learning. This approach is efficient for task execution because it is not always necessary to spend time on affordance learning if the available affordance knowledge is sufficient to solve the task. The proposed architecture enhances the robot's ability to solve complex real world tasks.

Samenvatting

De mogelijkheid om nieuwe dingen te leren is essentieel voor een robothulp om nuttige manipulatie vaardigheden te ontwikkelen en zo huishoudelijke taken op te lossen. Het is nuttig voor robots om de handelingsmogelijkheden (affordance) van objecten te leren om zo te kunnen voorspellen welke mogelijke effecten een handeling heeft. Deze informatie is taak-onafhankelijk en kan worden gebruikt voor het selecteren van acties om een verscheidenheid aan taken op te lossen. In dit proefschrift gaat onze interesse naar het efficiënt leren door robots en het hierbij gebruikmaken van handelingsmogelijkheden. Het leren van en het gebruikmaken van handelingsmogelijkheden zullen wij als een enkel proces beschouwen, in plaats van als twee verschillende fases.

Robots moeten kunnen omgaan met veranderende omgevingen wat vereist dat ze op elk moment handelingsmogelijkheden van objecten moeten kunnen leren. Bovendien moet rekening gehouden worden met de efficiëntie van het leerproces, want het is meestal een tijdrovend proces om de robot genoeg leerervaringen te laten opdoen door eigen handelingen. Een continue bereik van mogelijke bewegingen leid tot oneindig veel keuzemogelijkheden voor acties, hierdoor wordt het verzamelen van data nog moeilijker. Bovendien is het niet efficiënt om alle mogelijke objecten vanaf nul te leren gebruiken. De robot moet zijn eerdere ervaringen hergebruiken bij het leren gebruiken van nieuwe objecten.

Dit proefschrift ambieert om het leren en gebruiken van handelingsmogelijkheden efficiënter te maken. Wij hebben drie leermechanismes bestudeert die het verzamelen van data en oplossen van taken versnellen. Ten eerste, wij hebben voorgesteld om handelingsmogelijkheden te leren door altijd direct elke nieuwe succesvolle ervaring mee te gebruiken als leerervaring. Deze ervaringen kunnen ook gebruikt worden om het selecteren van de verkeerde actie te voorkomen in het zelfversterkende leermechanisme waar doelgerichte acties worden geformuleerd. Ten tweede stellen wij voor om actief handelingsmogelijkheden te leren, waarbij het opdoen van ervaringen word versnel d door actief nieuwe acties te verkennen. Modellen van handelingsmogelijkheden worden gebruikt om te voorspellen welke uitwerking een actie zal hebben en de nauwkeurigheid van de voorspelling wordt gebruikt als beloning voor het leermechanisme. Ten derde stellen wij voor om eerdere gebruikservaringen van relevante andere objecten te hergebruiken voor het sneller leren van vergelijkbare objecten. De robot beslist niet alleen zelf of het hergebruiken van ervaringen nodig is, maar ook welke aanpassingen aan het actie selectie mechanisme nodig zijn. Wij hebben laten zien, door experimenten met NAO robots, dat de voorgestelde methodes om handelingsmogelijkheden te leren efficiënter zijn dan eerder beschreven aanpakken.

Tot slot hebben wij een voorstel gedaan voor een agent-gebaseerde robot aansturing,

die het leren van handelingsmogelijkheden faciliteert en op meerdere cognitieve niveaus redeneert. Wij maken gebruik van redeneermechanismes op hoog abstractie niveau, in tegenstelling tot het leren van handelingsmogelijkheden op laag abstractieniveau door middel van fysieke interacties. De interactie tussen de abstractie niveaus wordt geregeld door de agent-gebaseerde aansturing. Hoewel de handelingsmogelijkheden worden opgeslagen op het hoge abstractie niveau, zijn deze gekoppeld aan fysieke bewegingen die worden opgeslagen op laag abstractie niveau. De agent kan autonoom besluiten om het leermechanisme aan of uit te zetten. Deze aanpak is efficiënt omdat het niet altijd nodig is om hier tijd in te steken als er al genoeg eerdere ervaringen zijn om de taak op te kunnen lossen. De voorgestelde aanpak verbetert de mogelijkheden van de robot om ingewikkelde taken op te lossen in de echte wereld.

Curriculum Vitæ

03-04-1985	Born in Wuhan, Hubei Province, China.
2000-2003	No.1 Middle School Affiliated to Central China Normal University.
2003-2007	University of Science and Technology of China, Bachelor of Science.
2007-2010	National University of Defense Technology, Master of Science.
2010-2015	Delft University of Technology, PhD researcher.
2015-	National University of Defense Technology, Lecturer.