Relational Affordances for Multiple-Object Manipulation

Bogdan Moldovan[a], Plinio Moreno[b], Davide Nitti[a], José Santos-Victor[b], Luc De Raedt[a]

[a]*Department of Computer Science, Katholieke Universiteit Leuven, Belgium*
[b]*Institute for Systems and Robotics (ISR/IST), LARSyS, Instituto Superior Técnico, Univ Lisboa, Portugal*

Abstract

The concept of affordances has been used in robotics to model action opportunities of a robot and as a basis for making decisions involving objects. Affordances capture the interdependencies between the objects and their properties, the executed actions on those objects, and the effects of those respective actions. However, existing affordance models cannot cope with multiple objects that may interact during action execution. Our approach is unique in that possesses the following four characteristics.

First, our model employs recent advances in probabilistic programming to learn affordance models that take into account (spatial) relations between different objects, such as relative distances. Two-object interaction models are first learned from the robot interacting with the world in a behavioural exploration stage, and are then employed in worlds with an arbitrary number of objects. The model thus generalizes over both the number of and the particular objects used in the exploration stage, and it also effectively deals with uncertainty.

Secondly, rather than using a (discrete) action repertoire, the actions are parametrised according to the motor capabilities of the robot, which allows to model and achieve goals at several levels of complexity. It also supports a two-arm parametrised action.

Thirdly, the relational affordance model represents the state of the world using both discrete (action and object features) and continuous (effects) random variables. The effects follow a multivariate Gaussian distribution with the correlated discrete variables (actions and object properties).

Fourthly, the learned model can be employed on planning for high-level goals that closely correspond to goals formulated in natural language. The goals are specified by means of (spatial) relations between the objects.

The model is evaluated in real experiments using an iCub robot given a series of such planning goals of increasing difficulty.

Keywords: affordances, relational affordances, probabilistic programming, object manipulation, planning

1. Introduction

The goal of robotics is to develop mobile, physical agents capable of reasoning, learning and manipulating their environment. Firstly, to achieve this, the physical agents need to deal with *uncertainty* in their physical world, including interpreting data from noisy sensors, processing image streams from cameras, and controlling noisy physical actuators for manipulation. To model this uncertainty, the use of probabilistic techniques is widespread in robotics [63], yet mostly without employing rich logical representations. Secondly, the physical agents need to deal with higher level knowledge for reasoning and planning in their environment. For this purpose, early approaches, such as the well-known Shakey robot, used the logical STRIPS representation [15], and several related symbolic representations [22, 61] have been used later. To model both uncertainty and high-level action knowledge, we shall employ representations from *statistical relational learning* (SRL) [18, 12, 11] and *probabilistic programming languages* (PPLs) [13], which combine expressive logical representations or programs with probabilistic reasoning and machine learning. Their use has recently been explored in robotics and their effectiveness has been shown in a kitchen scenario [24].

Recent advances in robotics have also led to increased capabilities of advanced humanoid robots, such as the PR2, iCub, NAO, and other robotic platforms. One characteristic of these humanoid robots is having two symmetrical arms with which they can manipulate their environment. The goal for humanoid robots is to manipulate objects in a household environment, such as a kitchen or a living room. In these cases, the robot should also consider background knowledge about the environment and objects to be manipulated, knowledge which a human would posses about the environment and task. For example, objects that are close together are likely to interact during manipulation, or several given types of objects are used together to achieve a task (e.g., a fork and a knife). Probabilistic programming (PP) is well suited to model background knowledge with the help of logical rules, while symmetries and generalisations are easily handled by the use of logic variables. Finally, PP can be used to model the uncertainties in the task, sensing, or actuators, with the help of probabilities.

Another recent and promising approach for the development of humanoid robots' skills has been the learning of *object affordances*. Affordances model the robot-world interaction by capturing *action opportunities* to structure the robot's environment (i.e., what can the robot do with an object?). For this, an affordance model captures the relationships between three variables: object properties, actions, and effects [38, 47]. Affordances have been used in various settings, from the modelling of relations between the numerical values of the object properties, actions, and effects [47], to the modelling of the effects of tool use [60, 58]. The concept of affordances follows the developmental robotics framework, which proposes acquiring new skills on top of old ones by experimentation and interaction with the environment [40].

Our goal is for a humanoid robot to be able to achieve a table-top object manipulation and arrangement task, involving multiple manipulation actions on various objects. We will make use of relational affordances[46, 45], a recent extension of the concept of affordances with the help of PP, which allows to model interactions and spatial relations between *multiple* objects in the environment. Our relational affordances is defines a joint probability distribution of the object properties, actions, and effects, considering: (i)Actions executed on only one object and (ii) actions executed on pairs of objects. Then, we add rules that allow to apply the joint distribution in scenarios with any number of objects. Thus the robot will be able to use this model to do probabilistic inference in a multiple object scenario, given his observations in the environment in order to achieve its task.

The distinguishing feature of our approach is the use of probabilistic programming, as opposed to previous approaches using Bayesian Networks [38, 47], which cannot capture the relationships amongst multiple objects. Using PP has the advantage that less data is needed for training, and that PP models can generalise to any number of objects in the scene without the need for retraining. Through interaction with the environment, the robot will learn a relational affordance model for its basic actions. The model for single-arm actions will be transferred to the other arm similar to our previous work [44], which eliminates the re-training phase of the other arm. We also consider the simultaneous two-arm pushing action that can produce similar effects independently of the size of the object. Once this model is learnt, the robot is given a high-level object arrangement task in terms of relations between objects (e.g., place all the cylinders to the left of the cubes). The task can be achieved by a sequence of actions, each of them involving any of the two hands of the robot. The robot will need to create a plan of basic actions that can achieve the goal. We will illustrate the approach with an iCub robot in a table-top setting.

1.1. Approach and Contributions

The main contribution of this article is a relational affordance model that considers continuous and discrete random variables for modelling the state of the world. The states and the transitions between them are modeled by a joint probability distribution over the actions, their effects and the objects. The main tool used to define the hybrid state distribution is the Distributional Clauses (DCs) formalism [21, 53], a PPL that integrates logic programming and probability theory. More specifically, we shall use an extension of DCs, Dynamic DCs (DDCs) together with the DC Particle Filter (DCPF) [54, 52, 53] for reasoning and planning. Compared to previous papers on relational affordances [46, 45, 44], the contribution of this paper is the first affordances model that (i) considers relations and interactions between multiple objects, (ii) represents the state of the world using discrete and continuous random variables, (iii) parametrizes the actions according to the robot capabilities, and (iv) employs sequential planning algorithms working with these hybrid representations. The following paragraphs detail the important aspects of our approach.

In our previous work, the main contribution was: (i) To introduce the relational affordances [46], which can model multi-object scenes with a variable number of objects by generalising a model trained with two-object interactions using probabilistic programming. Relational affordances allow for modelling actions and effects based also on the environment and relationships with other objects (e.g., a fork affords different actions if on the table or on the ground), and for modelling action effects on other objects in the environment with which the object acted upon might interact (e.g., picking up that fork might also move the knife next to it). As opposed to the previous methods of modelling affordances with BNs, the use of a PP model works for any number of objects in the scene, while also providing increased model comprehensibility. In [44] we extended the notion of a relational affordance to a hybrid one and realized (ii): the interactions between continuous and discrete random variables were learned using a Linear Continuous Gaussian Bayesian Network (LCG), and the learned model was then generalised to other environments using the DCs.

In our work on relational affordances [46, 44], the action repertoire of the robot was limited to actions having only one parameter. In the present paper, we expand the action repertoire by adding a parameter to each action to realize (iii), which enlarges the possible number of setups, allowing the robot to perform more tasks than before. In addition, we enlarge the repertoire of the actions by transferring a single-arm model of its symmetric counterpart, and add a simultaneous two-arm pushing action that causes similar effects as single-arm pushes, independently of the size of the object.

Other work on relational affordances [45, 44] have extended the previous models to more complex table-top manipulation tasks. This includes learning and using sequences of actions learnt by imitation, and extending the model to two-arm use by the robot in settings where two-arm actions can be approximated by combinations of single-arm actions in the discrete case. The models could use additional background knowledge in the form of logical rules in order to better model the given manipulation task. As opposed to [45] we will present a planning algorithm that represents the state of the world as a mix of continuous and discrete random variables (ii), being able to deal better with uncertainty during execution because it considers the effects as continuous random variables. (iv) The goal presented to the robot will be a high-level goal composed of spatial relations between the objects. This is in accordance with human-robot interaction approaches and represents a more realistic goal a human would ask from a robot, which has a more compact representation and could be easily explained in natural language text. For example, a human would ask the robot to place a cup near the plate to the right of the glass, rather than define a goal based on specific x-y coordinate locations for all the objects in the scene.

The system was be tested with a real iCub robot with its perception used to detect objects, and with its available motor skills (while the model learning will be done in the iCub simulator).

The full object arrangement planning task we tackle is based on previous research on relational affordances [46, 45, 44], extended with parameterised actions (iii), high-level goals (iv), and a planning algorithm (iv), in order to fully model a table-top task for a robot.

1.2. Structure of the Paper

This paper is organized as follows: Section 2 presents background information and related work, Section 3 the relational affordances for manipulation, and in Section 4 we describe our approach for learning a relational affordance model from exploration data. In Section 6 we describe our approach for using the affordances model as the state of the world for sequential planning. Finally, Section 7 presents experimental results, and we conclude in Section 9.

2. Background and related work

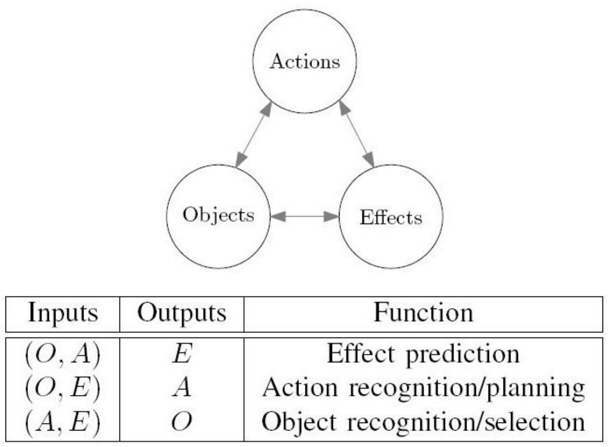
In this section we will present an overview of affordances and their relational extensions, followed by the necessary background on probabilistic programming languages, which will be used to build our models. Finally, we describe sequential planning from an artificial intelligence point of view and its application to robotic manipulation.

2.1. Affordances

Affordance models for robots are based on a concept introduced by J. J. Gibson [19]. Affordances define the relationships between the robot and the environment through the robot's available sensing and motor capabilities [38, 47]. Object affordances are learnt by a robot through an initial motor exploration phase, during which the robot manipulates objects in the environment, and perceives the effects of its actions. Although there are many different definitions of affordances in the literature, we view an affordances model as **a joint probability distribution over three sets of variables, representing objects, actions and effects**, as proposed previously [47]. Using an affordances model, given two of these variables, one can predict the third, allowing to perform three tasks: i) predict the *outcome* of an action (infer E, given O and A), ii) recognize a performed action (infer A, given E and O), or iii) select objects according to a task requirement (determine O, based on observed A and E) [47]. For example, affordances can be used for imitation learning [38] or action prediction by computing the *maximum a posteriori probability* (MAP) estimate $\arg\max_A P(A|O,E) = \arg\max_A \frac{P(A,O,E)}{P(O,E)}$, given the values of O and observing the E. A generic affordance model and its three uses are shown in Figure 1.

Research on affordances ranges from object categorisation from human demonstration [28], to learning predictive features in affordance-based robotic perception systems [16], and to the learning and use of the traversability affordance using range images on a mobile robot [65]. Affordances have been used especially in the context of imitation learning [38, 47, 33]. However, these latter ones used Bayesian Networks to model affordances, which are tailored to a specific number of objects in the scene and cannot easily generalise to any number of objects.

Other work related to the present paper includes research on tool use for robots [4], which learns the tool affordances of an object from a human demonstration together with a set of robot experimentations based on inductive logic



Inputs	Outputs	Function
(O, A)	E	Effect prediction
(O, E)	A	Action recognition/planning
(A, E)	O	Object recognition/selection

Figure 1: Affordances: relations between objects, actions, effects [38, 47].

programming, and research on behaviour-grounded tool affordances [60, 58], which provides algorithms for the robot to learn the effects of its actions with given tools on other objects.

An overview of the several affordance formalisms and their relation to planning and robot control was presented in [10]. Among the latest research, there is affordance learning for multi-step planning [66], which tackles a related problem, but where the learnt affordance model is only for single objects single-arm actions, and where object relationships and interactions are not taken into account. Similar work on affordance frameworks [6, 59] have defined affordance formalisms for control and planning, but without involving learning from demonstrations or a generalised modeling of multiple object interactions.

Finally, this paper is in the same area as that of probabilistic robotics [63] and of providing robots with logic and probabilistic reasoning capabilities [24, 22].

2.2. Probabilistic Programming Languages

We will now introduce basic concepts of PPLs, and illustrate how they can be used to model our setting from Figure 2. To create a model in PPL, one writes a program that consists of a set of probabilistic facts and a set of logical rules (which express *domain knowledge*). Once the program defining the probabilistic relational model is designed, inference methods are available for computing the probabilities of a user's query.

In logic programming, an atomic formula or atom $pred(t_1, ..., t_n)$ consists of a predicate $pred/n$ of arity n and t_i terms. A term is either a (lowercase) *constant*, (uppercase) *variable*, or functor (function) $func/n$ applied on n terms. These can be used to represent our object properties O and effects E. For example, the atom $\texttt{shape}(\texttt{o1}, \texttt{square})$ represents the object property $shape(o1) = square$ from Figure 2, and $\texttt{touches}(\texttt{o1}, \texttt{o2})$ that the objects touch one another.

A *definite clause* is an expression of the form $h \leftarrow b_1, ..., b_n$, where h and b_i are atoms. It states that h is true whenever all b_i are true. For example, the

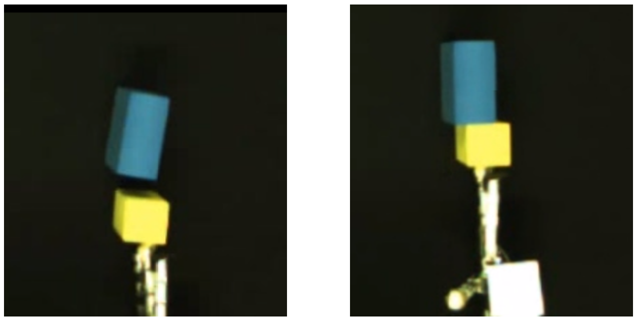


Figure 2: Table-top setting with two objects: (left): initial state, (right): final state after a push action.

clause:

$\texttt{displ}(\texttt{Obj}, \texttt{D}) \leftarrow \texttt{push}(\texttt{Obj}).$

states that for all objects pushed, the object will have a displacement of D (centimeters). The term `Obj` is a logical variable, that does not refer to a particular object, and can be substituted with any other term. Variables in clauses are assumed to be universally quantified.

A *substitution* $\theta = \{Y_1 = t_1, ..., Y_n = t_n\}$ maps each variable Y_i to a term t_i. Applying a substitution θ to an atom *pred* yields $pred\theta$, where each occurrence of Y_i in *pred* is replaced with t_i. For example, for $\theta = \{\texttt{Obj} = \texttt{o1}, \texttt{D} = \texttt{3}\}$, the above example becomes:

$\texttt{displ}(\texttt{o1}, \texttt{3}) \leftarrow \texttt{push}(\texttt{o1}).$

and states that `o1` is displaced by $3cm$ when pushed, as in Figure 2.

Distributional Clauses (DCs) [21, 53], an extension of the distribution semantics in [57], augment the traditional logic programming formalism to define random variables. DCs are a well suited PPL for our task since they allow one to define random variables with any distribution, continuous or discrete. The mix of continuous and discrete variables deals better with uncertainty than the purely discrete distribution from ProbLog, which was used in our previous affordances model [46].

A *distributional clause* is an expression of the form $h \sim \mathcal{D} \leftarrow b_1, ..., b_n$, where b_i are atoms and $\sim$ a binary predicate written in infix notation. Informally speaking, whenever the conditions in the body $\texttt{b}_1, \ldots, \texttt{b}_\texttt{n}$ hold, a random variable `h` is defined with distribution $\mathcal{D}$. A distributional clause is a powerful template to define conditional probabilities, indeed $\texttt{b}_\texttt{i}$, `h`, and $\mathcal{D}$ can contain logical variables that parameterise the clause. Formally, in a DC, each ground instance of the clause $(h \sim \mathcal{D} \leftarrow b_1, ..., b_n)\theta$, for a substitution θ, defines the random variable $h\theta$ being distributed according to distribution $\mathcal{D}\theta$ when all $b_i\theta$ hold. For example, we can model that all pushed objects have a displacement represented by a Gaussian distribution with mean $3cm$ and variance $1cm$:

$$\mathtt{displ(Obj)} \sim \mathtt{gaussian}(3,1) \leftarrow \mathtt{push(Obj)}.$$

A term $\simeq(d)$ constructed from the reserved functor $\simeq/1$ represents the value of the random variable d.

Once the model is defined, inference algorithms based on sampling [21, 54] can be used to compute the probability of a query. For example, one can compute the probability of the displacement of $o1$ from Figure 2 being greater than $6cm$: $P(\simeq(\mathtt{displ(o1)}) > 6)$.

The procedure used to generate possible worlds of a DC program, defines the semantics and a basic inference algorithm [21]. A possible world is generated starting from the empty partial world $x = \emptyset$; then for each distributional clause $\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{b_1}, \ldots, \mathtt{b_n}$, whenever the body $\{\mathtt{b_1}\theta, \ldots, \mathtt{b_n}\theta\}$ is true in the set x for the substitution θ, a value v for the random variable $\mathtt{h}\theta$ is sampled from the distribution $\mathcal{D}\theta$ and $\mathtt{h}\theta = v$ is added to the new partial world $\hat{x}$. This is also performed for deterministic clauses, adding ground atoms to $\hat{x}$ whenever the body is true. This process is then recursively repeated until a fixpoint is reached, that is, until no more variables can be sampled and added to the world. This sampling procedure can be viewed as a generalization of ancestral sampling for Bayesian networks, in which a random variable is sampled when its parents are already sampled.

Sampling full worlds is generally inefficient. More efficient sampling algorithms [21, 53] have been proposed to sample only those random variables that are relevant for answering the query.

In the context of temporal domains and sequential planning, we will use Dynamic Distributional Clauses (DDCs) [54, 52, 53], which are an extension of DCs for temporal domains. DDCs define a discrete-time stochastic process following the same idea of a Dynamic Bayesian Network [50], by using sets of clauses that define:

1. the prior distribution: $\{\mathtt{h_0} \sim \mathcal{D} \leftarrow \mathtt{body_0}\}$,
2. the state transition model: $\{\mathtt{h_{t+1}} \sim \mathcal{D} \leftarrow \mathtt{body_t}\}$,
3. the measurement model: $\{\mathtt{z_{t+1}} \sim \mathcal{D} \leftarrow \mathtt{body_{t+1}}\}$,
4. a random variable at time t from other variables at the same time: $\{\mathtt{h_t} \sim \mathcal{D} \leftarrow \mathtt{body_t}\}$.

For example, to model an environment with balls, where the next position of every ball is equal to the current position plus some Gaussian noise, one can write:

$$\mathtt{pos(Obj)_{t+1}} \sim \mathtt{gaussian}(\simeq(\mathtt{pos(Obj)_t}), \mathtt{cov}) \leftarrow \mathtt{ball(Obj)}.$$

This signifies that the position of the object at time $t+1$ is given by a Gaussian distribution with mean equal to the position of the object at time t, and covariance *cov*.

Note that if no DDC clauses that define the random variable $\mathtt{h_{t+1}}$ apply for a given state and action, $\mathtt{h_{t+1}}$ is undefined. This is useful to describe negative effects, e.g. a fact that is no longer true[1], or when a random variable is no longer needed in the next state. For example, if we care only about the objects on a table for manipulation, we can ignore and thus remove from the next state all the objects that fall off the table or that are grabbed by a human.

Given a set of DDC clauses, the Distributional Clauses Particle Filter (DCPF) [54, 53] can be used to perform filtering inference. Filtering, or state estimation, computes the probability density function $P(x_t|z_{1:t}, a_{1:t})$, where x_t is the current state, $z_{1:t}$ is the set of observations, and $a_{1:t}$ the actions (inputs) performed from time step 1 to t.

2.3. Planning

Even though the focus of the current paper is on affordances, our work is also related to planning and learning to plan. We first discuss probabilistic planning using MDPs, then its extension towards probabilistic STRIPS like representations, and finally some more specific approaches tailored towards robotics.

2.3.1. Probabilistic planning

Probabilistic planning is generally performed by solving a Markov decision process (MDP) for fully-observable problems or a POMDP for partially-observable problems. An MDP [69, 62] consists of a set S of states, a set A of actions the agent can take, a state transition model T from time t to time $t+1$, $T : S \times A \times S \rightarrow [0, 1]$ where $T(s_t, a_t, s_{t+1}) = p(s_{t+1}|s_t, a_t)$, and a reward function $R : S \times A \rightarrow \mathbb{R}$ that assign a reward $r(s, a)$ when the agent is at state s and performs action a. In our case, the state is relational, that is the state s is a set of ground relational atoms and/or pairs (variable, value). The goal of the agent is to maximize the expected (discounted) reward $E[\sum_{t=0}^{d} \gamma^t r_t]$. If d is finite we have a finite horizon MDP otherwise an infinite horizon MDP. The term $\gamma \in [0, 1]$ is a discount factor needed to keep the sum bounded for infinite horizon MDPs, thus generally $\gamma = 1$ is used for finite horizon MDPs.

To maximize the expected reward, we need to find a (deterministic) policy π that assigns for each state s and time t the action to perform. For infinite horizon MDPs we can use stationary policies $\pi : S \rightarrow A$ that do not depend on the time t without loss of generality. While, a stochastic policy $\pi : S \times A \rightarrow [0, 1]$ assigns a distribution over actions $p(a|s)$ for each state s.

The expected reward starting from state $s_t = s$ and following a policy π is called value function (or V function): $V_d^\pi(s) = E[\sum_{k=0}^{d} r_{t+k}|s_t = s, \pi]$ for finite horizon MDPs (assuming $\gamma = 1$) and $V^\pi(s) = E[\sum_{k=0}^{\infty} \gamma^k r_{t+k}|s_t = s, \pi]$ for infinite horizon MDPs. In the finite case the V function depends on the

[1]we assume the close-world assumption: anything that is undefined is considered false

number d of remaining steps, this is not the case for infinite horizon MDPs. The expected reward starting from state $s_t = s$ executing action $A_t = a$ and following a policy π is called action-value function (or Q function). For finite horizon MDPs: $Q_d^\pi(s, a) = E[\sum_{k=0}^{d} r_{t+k} | s_t = s, A_t = a, \pi]$ and for infinite horizon MDPs: $Q^\pi(s, a) = E[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, A_t = a, \pi]$.

An optimal policy π^* is a policy that maximises the V function for all states. The goal of planning goal is to obtain such a policy.

Our state representation involves both discrete and continuous variables, where the optimal V function and policy cannot be computed exactly in the general case. In [72], a propositional exception is proposed for a deterministic transition model at least for the continuous part. For more general cases approximations are needed, such as Monte-Carlo methods.

The class of planning algorithms closest to our needs are sampled-based planners that use Monte-Carlo methods to estimate the optimal Q/V function. Some of the more notable examples include ε-soft on-policy Monte Carlo control [62], sparse sampling [26], UCT (Bandit based Monte-Carlo planning) [29], and PRADA for noisy probabilistic relational rules [36]. In Section 5.2, we shall introduce a simple probabilistic planner that suffices for our aims even though we have recently developed an extension of that planner for use with DDC [51].

2.3.2. Planning with STRIPS

For dealing with relations and objects in planning, one typically employs extensions of STRIPS like representation [15], such as Planning Domain Definition Language (PDDL) [41, 71]. The key idea is that actions are described using a pre-condition and the effects of the actions using an add- and delete-list. STRIPS and PDDL have been extended towards a probabilistic setting.

The representation that we shall use in Section 5 for modeling the state-transitions and for planning will be based on DDC rather than probabilistic STRIPS. We shall show that it is easy to map probabilistic STRIPS into the DDC representation. There are also some more subtle differences between the two representations. The DDC formalism does not make an explicit frame assumption, i.e. it is not assumed that what is not explicitly mentioned in the effects does not change. Another difference is that our affordances explicitly model a delta change between the two states, cf. also Figure 8 and Table 1.

Starting with Buridan [34] and Graphplan [3], many planners for probabilistic STRIPS have been developed, some of which are relevant to the present paper. For instance, some works on noisy indeterministic deictic rules are also concerned with planning and learning STRIPS rules from data or from noisy observations [55, 73], and some are applied on planning for object manipulation in a relational domain [37, 64]. Other related work focuses on learning action effects in partially observable domains [48]. However, all these related works are not done within an affordance framework that can learn and generalise from a robot motor exploration stage so they do not generalise from two-object interactions to a multiple object environment, or model the effects of actions performed with the other arm of a two-arm robot by symmetry. They also do not map demonstrations to the action space of a robot and are rarely performed in a real-world robotics setting that requires dealing with continuous distributions.

More related research in the context of planning is [35] and action object complexes (OACs) [70, 32] which have been proposed for enabling efficient planning and execution of actions at all levels of the cognitive architecture, by combining STRIPS rules with the concept of affordances. Other research in this direction includes [17] and [31], but this formalism doesn't involve the generalisation of multiple object interactions obtained from exploratory actions as done by relational affordances.

Some approaches in decision-theoretic planning tackle similar problem domains [68], but usually work with predefined and full action models, whereas we learn the affordance models. Finally, there is further research concerned with grounding planning operators by affordances [39] or an affordance formalism for planning from the perspective of user interfaces [1] among others.

2.3.3. Planning in Robotics Manipulation

Research on two-arm robot manipulation includes research on learning, representing and generalising a task which presents a programming-by-demonstration framework for extracting and generalising knowledge about a given task [7], and similarly a programming-by-demonstration framework for dual-arm manipulation tasks [74]. There is also research on motion planning for dual-arm manipulation and re-grasping tasks [67]. However, these do not use the concept of affordances, model or generalise over relations and interactions between manipulated objects and other objects in the environment, or build a two-arm manipulation model generalised from environment experimentation and the use of background knowledge.

Related to a table-top setting with multiple interacting objects, there is work on detecting [30] and manipulating [2, 20, 25] objects in cluttered environments, but this is usually concerned with detecting the objects and motion-planning for the arm in order to perform a grasp [20, 25], rather than creating a plan in order to solve a given task.

A related task as the one presented in this paper, planning push actions for object placement on a cluttered table surface, is performed in simulation [9] and with a PR-2 robot [14]. However, in these cases the object interactions are determined by a dynamics simulator based on the object physical properties, not taking uncertainty into account. This work does not generalise from two-object interactions to learn a model of a multiple-object setting.

3. Relational Affordances for Manipulation

In this section we describe the basic skills of the robot, an overview of the problem we address: The relational affordances model and its extension to sequential planning.

3.1. Basic Skills of the Robot

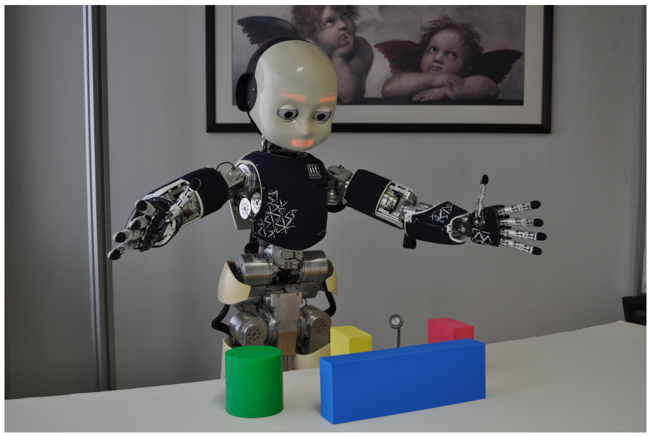
We employ, both in a real setting and in simulation, the *iCub humanoid robot* [43], which has a head with two cameras, two arms and two legs. The legs of the robot are immobile, and we use both arms, and both cameras. Each arm has a force-torque sensor that provides the observations for an impedance arm controller (i.e. from the shoulder joint to the wrist joint). However, the finger joints can be controlled only in position mode. The cameras allow to obtain the disparity map using stereo algorithms.

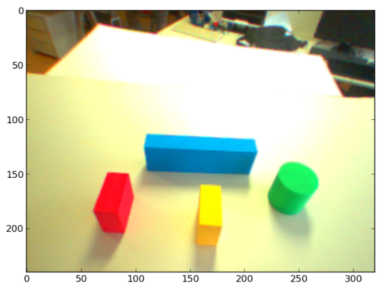
We assume the robot is provided with a set of core motor actions and perceptual skills. The motor actions are based on a Cartesian controller [56], which allows the hand to move between points in the space considering constraints in hand orientation and forces being exerted. The perceptual skills are based on color segmentation in images and stereo vision algorithms that allow to locate objects and their respective sizes. We build on these elements the basic skills of the robot: motor skills to perform the actions and perceptual skills to measure object features and effects.

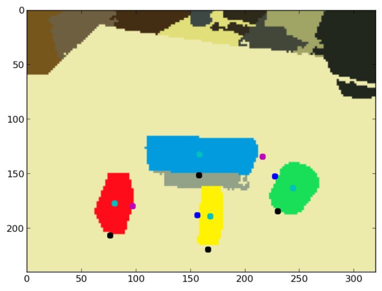
Motor actions tap and push are parameterised with the arm (left, right), the distance in *cm* and the direction that the hand moves over. The two arm push action is parametrised with the distance in *cm* that the hand moves over and the size of the object. The distance is relative to the current location of the objects, while the direction of the actions could be from left to right, right to left and nearer to further. The location of landmark points of the objects (see examples in Fig 3) provides the initial points in space for the actions, and each point is associated to the direction of the action. A point on the left side of the object will be the initial point for left-right direction, the bottom point of an object will be associated to nearer-further direction and so on. Figure 3 illustrates the detected action points on the segmented image of four objects. According to the position of the object's centroid relative to the robot, the right and left landmark points were either retrieved from the image or estimated from the other points. If the centroid was on the left side of the robot, the right point was retrieved and the left one estimated. Similarly, if the center was on the right side, the left point was retrieved and the right one estimated. The bottom points are always retrieved from the image.

The action execution procedure is provided with a force trigger, which is activated when the magnitude of the force is above a previously defined threshold. The motion of the arms is performed by a minimum-jerk Cartesian controller which reaches a position as close as possible to a given target position while coping with the kinematic and dynamic constraints of the iCub [56]. Figure 4 illustrates the object locations before and after every action execution by the real iCub.

Regarding visual perception, we assume object identification is provided by color segmentation, and object size is provided by stereo vision. Color segmentation is based on an algorithm relying on a synergistic approach combining a confidence-based edge detector and mean shift segmentation [8]. The image segmentation algorithm is applied on both cameras in order to find the enclosing


(a) Robot and objects





(b) Raw image (c) Segmented image with landmark points

Figure 3: Illustration of the table-top scenario for the real iCub, with its correspondent point of view of the robot and the segmentation result. The coloured points are associated to points as follows: cyan to centroids, black to bottom, magenta to right and blue to left.

region of objects on each image. Then, the centroid of the segmented region is extracted on both cameras in order to perform stereo triangulation. This process provides the 3D position of the object's centroid, which represents its location.

Object size is computed from the combination of segmentation and stereo algorithms. From the segmented image of one of the eyes, we extract the ellipse that encloses each coloured region [2]. The points in the ellipse that intersect its major axis are mapped onto the stereo disparity image for 3D perception. The distance between the 3D mapped ellipse points provide the object size.

[2]having the same normalized second central moments

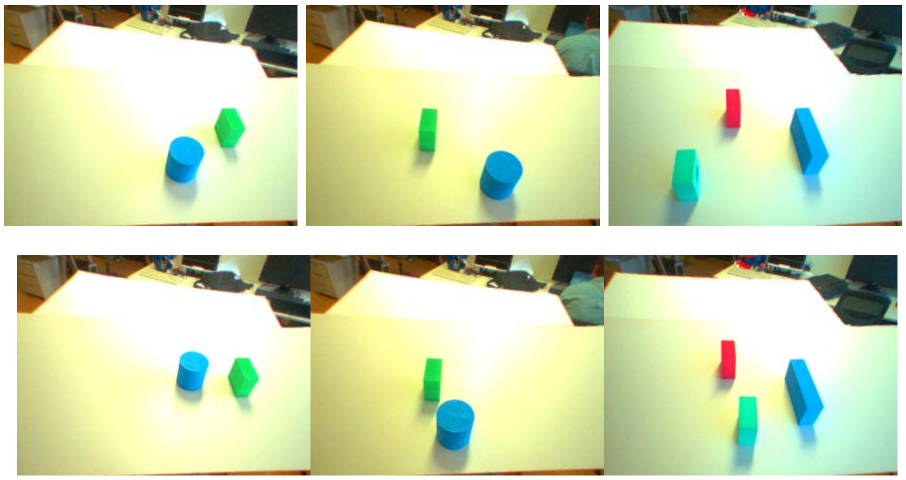

(a) Push (Left arm, 15cm) (b) Tap (Right arm, 10cm) (c) Tap (Left arm, 20 cm)

Figure 4: Action execution examples from the iCub's left camera point of view. The top row images show the location of the objects before the action execution and the bottom row images show their locations after action execution. Each column represents a different action

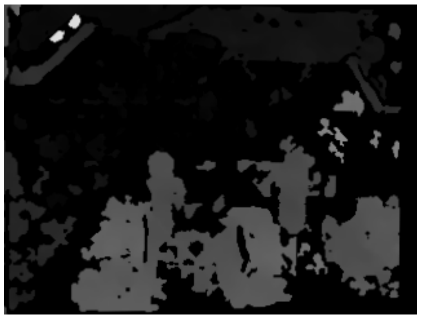
3.2. Problem Statement and Approach

We tackle a table-top scenario where a two-arm robot needs to manipulate multiple objects, that can interact with one another, in order to reach a given goal. To reach the goal, the robot will need to execute a sequence of its basic actions, whose relational affordance model it previously learns during a behavioural exploration stage.

One example of such setting can be seen in Figure 6. The robot will use its perception to detect the initial setting of the objects, as in Figure 6 (left). The robot is then given a high-level goal, specified as spatial relations between the objects in the setting. In this case, the goal is to place the long prisms to the left of the small prisms (as seen from the robot's point of view), while all objects need to be "in the shelf" (considered at the back of the table, behind the dashed line). One possible goal configuration the robot can reach can be seen in Figure 6 (right). To achieve this task, the robot first needs to: (1) tap the red object with the right arm (towards the left), then (2) tap the magenta object with the left arm (towards the right), and finally (3) push the magenta object with any of its arms. Note for example that in step (2) the object is only reachable with the left arm, and after it is tapped it is in a position where it can be acted upon with both arms.

Tackling such object manipulation scenario is composed of three different tasks, shown in Figure 7:

- Task (i) is learning a two-arm continuous relational affordance model: *given:* a) a set of corresponding O, A, E values collected from exploratory one-arm and simultaneous two-arm action executions in two-object envi-


(a) Disparity image

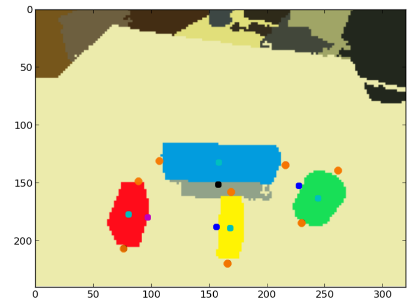

(b) Segmented image with landmark points

Figure 5: Illustration of the object size computation. Left-hand image shows the disparity map of the example shown in Figure 3. The orange points in the right-hand image show the points that intersect with the ellipse's major axis. The orange points are mapped onto 3D using their associated disparity value, and the 3D distance between each pair is defined as the object size.

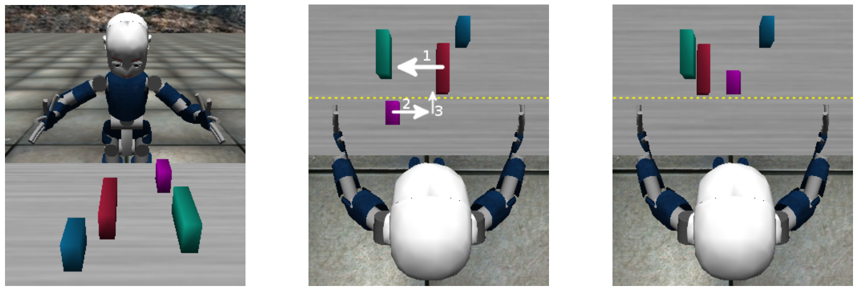


Figure 6: Table-top scenario with sequence of arm actions for object placement: (left): initial setting, (middle): actions to reach goal, (right): possible goal arrangement.

ronments, and b) background information about symmetries of left and right arm actions, *find:* c) a continuous setting relational affordance model of two-arm actions, modelled in a PPL.

- Task (ii) is modelling a state transition model: *given:* a) the relational affordances model learnt in Task (i), and b) a set of task constraints rules for determining applicable actions (e.g., do not act on potentially occluded objects), *find:* c) a state transition model for the robot actions, to be used for planning tasks.

- Task (iii) is the planning task used to evaluate our model: *given:* a) an initial scene from which using its perception the robot extracts the set of object properties values O, and b) a target goal, given as a set of spatial relations between the objects, together with, c) the state transition model for the robot actions obtained by Task (ii), *find:* d) the next best action to execute towards reaching the goal.

The robot can then repeat the next best action inference and action execution until the goal is reached, or until we reached a predefined maximum number of actions.

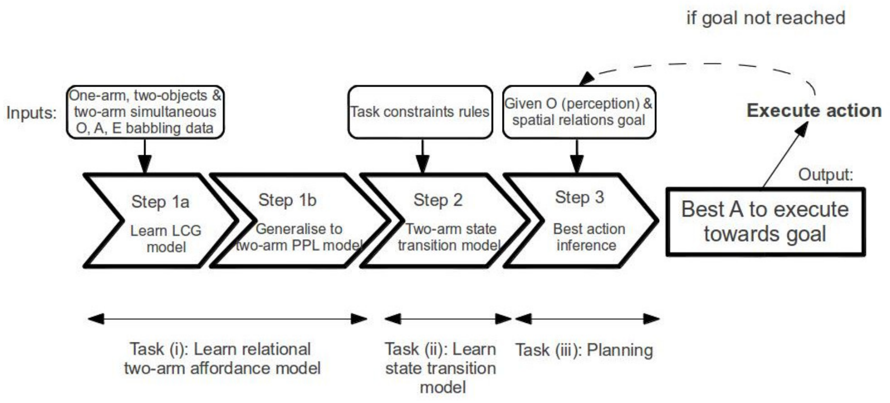

if goal not reached
Inputs:
One-arm, two-objects & two-arm simultaneous O, A, E babbling data
Task constraints rules
Given O (perception) & spatial relations goal
Execute action
Output:
Step 1a Learn LCG model
Step 1b Generalise to two-arm PPL model
Step 2 Two-arm state transition model
Step 3 Best action inference
Best A to execute towards goal
Task (i): Learn relational two-arm affordance model
Task (ii): Learn state transition model
Task (iii): Planning

Figure 7: Pipeline for table-top two-arm object manipulation.

To solve these tasks, several steps are required, as shown in Figure 7:

- 1a) learn a Linear Continuous Gaussian (LCG) Bayesian Network (BN) from single arm and simultaneous two-arm exploratory data,
- 1b) from the LCG model, build the two-arm continuous domain relational affordances model in a PPL,
- 2) build a state transition model from the relational affordances model, and
- 3) infer best action to execute to reach goal (step repeated until goal reached).

Thus, the planner will use the low level information it acquires from its sensors, and will employ the state transition model obtained from the previously learnt relational affordances model together with the set of given background rules about its actions.

3.3. Relational affordances

Affordances are modeled as relations between the following three variables: the set of *objects* and their *properties* as being detected by the robot sensors: $O = \{o_1, o_2 ..., o_n\}$, the repertoire of *actions* available to the robot, $A = \{a_1, a_2 ..., a_n\}$, and the *effects* of performing those actions $E = \{e_1, e_2 ..., e_n\}$ as detected by the sensors as changes in object features.

A relational affordance is an extension of the affordances model of Figure 1, where rather than using a propositional representation for object properties, actions, and effects, we now use a relational one. Thus a relational affordance is a *joint distribution* over a relational representation for O, A, and E.

More formally, the relational representation for O, A, E can be expressed as follows. Let $\mathcal{Z}$ represent the set of all objects in the environment that the affordances model, and an uppercase $\mathtt{Z}$, optionally followed by a subscript, a variable in the domain $\mathcal{Z}$. Then the *object properties* O are represented by the set of all random variable atoms of the form $\mathtt{o_i(Z_1, ..., Z_{m_i})}$ (e.g., $\mathtt{distance(ball, book)}$). The set of *actions* on object Z is denoted by: $A(Z)$, and the *effects* E are the set of all random variable atoms $\mathtt{e_i(Z_1, ..., Z_{n_i})}$.

For example, a relational affordance could represent a joint probability distribution that could specify that :

$$\begin{aligned}&\mathtt{P(shape(ball) = sphere, distance(ball, book) > 3,}\\&\quad\mathtt{A(ball) = tap(ball), rel\_displacement(ball, book) > 0) = 0.58}\end{aligned}$$

To define the joint probability distribution $P(O, A, E)$ we will use a relational representation in the form of a PPL program.

3.4. Planning with Relational Affordances

We will now describe our concept of relational affordances for solving table-top planning tasks. An example of such setting with two objects can be seen in Figure 2. The general relational affordances formalism defined a joint probability distribution over O, A, E. In order to be able to tackle a temporal domain and a planning task, we will additionally define a state transition model and action representation. We will illustrate this with a DDCs-style syntax.

A state represents a description of the environment of the robot at a given time. A state is a conjunction of all grounded atoms $\mathtt{o_i(Z)}$ for all the objects in the domain $\mathcal{Z}$.

For example, in the table-top setting from Figure 2, considering just the object properties $\mathtt{shape}$ and $\mathtt{distance}$, we could have the following state at time t:

$$\mathtt{shape(o1)_t = square, shape(o2)_t = rect, distance(o1, o2)_t = 5.}$$

We will define actions, their preconditions and their effects using dynamic distributional clauses.

The *preconditions pre* are a set of relations on the random variable atoms $\mathtt{o_i(Z)}$ from O. An action $A(Z)$ can be executed in the current state S_t only if there is a substitution θ for the variables in *pre* such that *pre*θ holds in S_t. As an example, consider an action $\mathtt{push(Z_1)}$, whose preconditions can be:

$$\begin{aligned}&\mathtt{pre(push(Z_1)_t)} \leftarrow \simeq\mathtt{(shape(Z_1)_t) = square, object(Z_2), Z_2 \neq Z_1,}\\&\quad\simeq\mathtt{(distance(Z_1, Z_2)_t) < 5.}\end{aligned}$$

where $\mathtt{object(Z_2)}$ holds if $\mathtt{Z_2}$ is an object in the scene. The preconditions hold for the substitution: $\theta = \{Z_1/o_1, Z_2/o_2\}$.

Furthermore, in this paper we will refer to an action as applicable in the current state if it can be executed in the current state, thus if the preconditions hold. Note that the applicability refers to the action and the object involved, thus the same action (e.g., push) can be applicable for an object `o1` but not applicable for another object `o2`.

To define action applicability we use a set of (deterministic) DDC clauses of the form $\mathtt{applicable(a(Z)_t)} \leftarrow \mathtt{pre(a(Z)_t)}$. For example, `push` is applicable only in those states where the object is reachable by the robot's arm:

$$\mathtt{applicable(push(Obj))_t} \leftarrow \mathtt{reachable(Obj)_t}.$$

The *effect eff* describes the state change after the execution of the action $A(Z)$. In DDCs, this can be achieved by defining a state transition model. So, for each atom that will be defined in the next state at time $t+1$, we need to define a state transition model clause: $\mathtt{h_{t+1}} \sim \mathcal{D} \leftarrow \mathtt{body_t}$. The body of the clause contains the action and the preconditions that depend on the current state:

$$\mathtt{h_{t+1}} \sim \mathcal{D} \leftarrow \mathtt{A(Z)_t, pre(A(Z)_t)}.$$

For example, to specify the effect on displacement of push action on a square object from Figure 2, we can write:

$$\mathtt{displ(Obj)_{t+1}} \sim \mathtt{gaussian(3,1)} \leftarrow \mathtt{push(Obj)_t}, \\ \simeq\!(\mathtt{shape(Obj)_t}) = \mathtt{square}.$$

Note that compared to the rule effects eff, in our table-top setting affordance effects E represent relative changes in one or more object properties O due to the action. Thus the affordance effects can be seen as a delta change between the states S_t and S_{t+1}. In the following section we describe how to discover the relations between the variables of the relational affordances model: O, E and A.

4. Learning Relational Affordance Models

In this section we will describe the **learning of a two-arm continuous relational affordances model** from a behavioural exploration stage, which is **Task (i)** of our approach.

4.1. Affordances in Table-top Setting

We will define now the object properties (O), actions (A), and effects (E) that we will use in our model.

The **object properties** O are the following: *shape*, and *relations* relative distance along the x-axis (*distX*) and y-axis (*distY*) between two objects (in cm). All distances will be measured from the centroids of the objects, as observed from perception and determined by stereo triangulation of the centroids obtained from color segmentation. These object properties $distX$ and $distY$ are shown in Figure 8(l), with the objects' position before (l) and after (r) an action (tap) execution. The x and y-axes correspond with the iCub x and y-axes from the robot's viewpoint. To facilitate modelling in a continuous domain setting, we use a Cartesian coordinate system instead of a polar one as in [46, 45]. We will use five different objects of one of three different shapes. The shape of two of the objects is small prism (*sprism*) (sizes: $4cm \times 8cm \times 8cm$ and $4cm \times 9cm \times 7.5cm$). There are two big prisms (*bprism*) (sizes: $4cm \times 14.5cm \times 9cm$ and $4cm \times 16cm \times 8cm$). There is one long bar (*lbar*) (size: $27cm \times 6cm \times 5cm$).

The **action** A is one of two basic arm core motor actions: *tap* (right-to-left hand movement for the right arm, left-to-right for the left) and *push* (away movement for both arms). As an extension of previous affordances models [46], the action is parameterised by the distance in *cm* that the arm is moved from its pre-action position next to the object until the action is finished. The arm movement distance values can be one of: $10cm$ and $20cm$ for the *tap* actions, and $15cm$ and $25cm$ for the *push* actions (but more can be modelled as well). We will also have a two-arm simultaneous *push* action on the same object. Alternatively, this could also have been modelled by a combination of single-arm actions as in [44], but because of iCub limitations and to increase accuracy, the robot will directly learn from two-arm push exploration data.

The **effects** E correspond to differences in object attributes before and after the action is performed. We use the displacements along the x-axis ($displX$) and y-axis ($displY$) of the centroid of each object. These effects $displX$ and $displY$ are shown in Figure 8(r), which overlays the initial objects' positions over their final positions.

To learn an affordances model, the robot first performs a behavioural exploration stage, in which it explores the effect of its actions on the environment. For this behavioural exploration stage, for the single-arm actions the robot uses its right-arm only. For these actions a model of the left-arm will be later built by exploiting symmetry as in [44]. We include the simultaneous two-arm *push* on the same object in the exploration phase, allowing for a more accurate modelling of action effects for the iCub.[3]

The exploration phase consists of placing pairs of objects in front of the robot at various positions. The robot executes one of its actions A described above on one object (named: main object, O_{Main}). O_{Main} may interact with the other object (secondary object, O_{Sec}) causing it to also move. Figure 8 shows such a setting, with the objects' position before (l) and after (r) a right-arm action ($tap(10)$) execution.

During this behavioural exploration stage, data for O, A and E are collected

[3]As opposed to the two-arm affordances modelling in [44], we also include in the exploration phase the two-arm simultaneous actions whose effects might not always be well modelled by the sum of the individual single-arm actions.

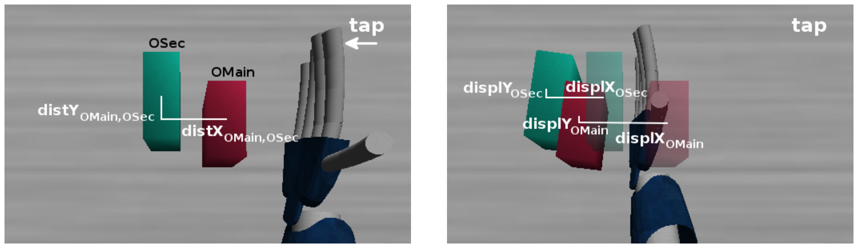


Figure 8: Relational O before (l), and E after the action execution (r).

for each of the robot's exploratory actions. The robot executed 150 such exploratory actions. One example of collected data during such an action is shown in Table 1. Note that these values are obtained by the robot from its perception, which naturally introduces uncertainty, which the relational affordances model takes into account (e.g., the displacement of O_{Main} is observed to be a bit more than $10cm$).

Table 1: Example collected O, A, E data for action in Figure 8

Object Properties	**Action**	**Effects**
$shape_{O_{Main}} : sprism$ $shape_{O_{Sec}} : sprism$ $distX_{O_{Main},O_{Sec}} : 6.94cm$ $distY_{O_{Main},O_{Sec}} : 1.90cm$	$tap(10)$	$displX_{O_{Main}} : 10.33cm$ $displY_{O_{Main}} : -0.68cm$ $displX_{O_{Sec}} : 7.43cm$ $displY_{O_{Sec}} : -1.31cm$

During the exploration phase, we also learn the action space of each action. As the iCub is not mobile, and each arm has a specific action range, each $a_i \in A$ can be performed when an object is located in a specific action space. An object can be acted upon by both arms, by one arm but not the other, or it can be completely out of the reach of the robot. If the exploratory arm action on an object fails because no inverse kinematics solution was found, then that object is not in that arm's action space. We will show later how any spatial constraints, such as action space, can be modelled with logical rules.

4.2. Learning the Model

The model will be learnt from the data collected during the robot's 150 exploratory actions, one instance of such data as illustrated in Table 1. We will model the (relational) object properties: *distX*, *distY* (the x and y-axis distance between the centroids of the two objects), and the effects: *displX* and *displY* (the x and y-axis displacement of an object) with continuous distribution random variables. We will start by learning a *Linear Conditional Gaussian (LCG) Bayesian Network* [27]. An LCG BN specifies a distribution over a mixture of discrete and continuous variables. In an LCG, a discrete random variable may have only discrete parents, while a continuous random variable may have both discrete and continuous parents. A continuous random variable (X) will have a single Gaussian distribution function whose mean depends linearly on the state of its continuous parent variables (Y) for each configuration of its discrete parent variables (U) [27]. This LCG distribution can be represented as: $P(X = x|Y = y, U = u) = \mathcal{N}(x|M(u) + W(u)^T y, \sigma^2(u))$, with M a table of mean values, W a table of regression (weight) coefficient vectors, and σ a table of variances (independent of Y) [27].

To learn an LCG BN for our setting, we will approximate $displX$, $displY$, and $distX$ and $distY$ by conditional Gaussian distributions over the short distances over which objects interact. These distances will be enforced by adding logical rules.

The LCG model of our setting is shown in Figure 9, where discrete random variables are represented by a single ellipse, and continuous ones by a double ellipse. $displX_{O_{Main}}$ and $displY_{O_{Main}}$ only depend on A and the object *shape* since the hand is moved over a fixed distance (with a given tolerance) for each parameterised action. $displX_{O_{Sec}}$ and $displY_{O_{Sec}}$ depend on both the relative distance O_{Sec} is away from O_{Main} and the shapes of both objects. According to the objects, static friction coefficients and masses, the structure of the LCG BN in Fig. 9 may be different. For instance, if the secondary object's mass is significantly larger that the main object's mass, the relational distance between objects ($\mathrm{distX_{OMain,OSec}}$ and $\mathrm{distY_{OMain,OSec}}$) may be linked to the displacement of the main object ($\mathrm{displX_{OMain}}$ and $\mathrm{displY_{OMain}}$). In the case of our experimental setup the displacement of the main object does not have the relational distance between the objects as parents because the arm action is strong enough and the objects light enough such that as the arm is pushed along over the specific distance, so is the main object which is dragged along the respective distance.

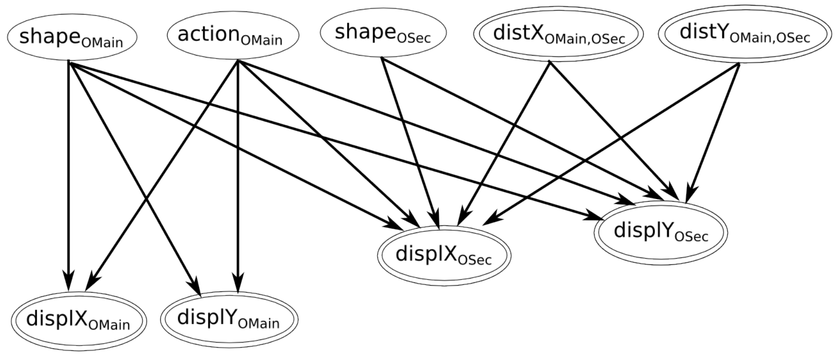


Figure 9: LCG BN model for two-object interaction (single ellipse = discrete random variable, double ellipse = continuous random variable)

The parameters of the LCG model are learnt from the collected exploration data (e.g., Table 1) by using the maximum likelihood parameter estimation from the BNT toolbox [49] for Matlab. For example, during our *tap(10)* action for two interacting cubes (as in Figure 8), the displacement of O_{Sec} on the x-axis is (in cm):

$$\mathcal{N}(7.05 + 0.57 * distX_{O_{Main},O_{Sec}} + 0.02 * distY_{O_{Main},O_{Sec}}, 0.41). \qquad (1)$$

This makes sense intuitively as the second cube is moved along by the $tap(10)$ action, so we expect its displacement to depend mainly on $distX$, but also a little bit on $distY$ if the objects are not aligned, as in Figure 8.

The obtained LCG BN models two-object interaction due to the robot's basic actions. Now we can model the LCG using DCs syntax (as introduced in Section 2), to generalise to a relational affordances model for a multiple object setting. For the modelling of affordances using DCs for our task, the main predicates we will use are presented in Table 2.

Table 2: Predicates used for affordance modelling

Predicate	Meaning
`shape(Obj, Shape)`	The shape of object `Obj` is `Shape`.
`distX(Obj1, Obj2)`	Distribution of the relative x-axis distance between objects `Obj1` and `Obj2`.
`distY(Obj1, Obj2)`	Distribution of the relative x-axis distance between objects `Obj1` and `Obj2`.
`displX(Obj)`	Distribution of the x-axis displacement of `Obj`.
`displY(Obj)`	Distribution of the y-axis displacement of `Obj`.
`tap(Obj, Arm, Param)`	A parameterised tap on object `Obj` where the arm `Arm` is moved a distance of `Param`. `Arm` can be `left` or `right`.
`push(Obj, Arm, Param)`	A parameterised push on object `Obj` where the arm `Arm` is moved a distance of `Param`.
`approx_ok(A, ObjM, ObjS, DX, DY)`	True if the Gaussian approximation for the action effect holds for the action `A` on main object `ObjM` and with secondary object `ObjS` when the x-axis distance between the objects is `DX` and the y-axis distance between the objects is `DY`.
`coordX(Obj)`	Distribution of the x-axis coordinate of object `Obj`.
`coordY(Obj)`	Distribution of the y-axis coordinate of object `Obj`.
`occluded(Obj)`	True if object `Obj` is potentially occluded behind any other object in the scene.
`left(Obj1, Obj2)`	True if object `Obj1` is to the left of object `Obj2`.
`inshelf(Obj1)`	True if the y-axis coordinate of the centre of object `Obj1` is greater than $40cm$.
`near(Obj1, Obj2)`	True if the centre-to-centre distance between objects `Obj1` and `Obj2` is less than $10cm$.

We generalise over the number of objects as in [46], by introducing variables for objects (e.g., $\texttt{displX}(\texttt{O}_{\texttt{Main}})$ for the $displX_{O_{Main}}$ in the LCG), and so build a general multiple object PPL model from the two-object LCG BN.

For example, to transform the LCG Equation 1 in DCs, one writes:

$\texttt{displX}(\texttt{O}_{\texttt{Sec}}) \sim \texttt{gaussian}(\texttt{Mu}, 0.41) \leftarrow \texttt{tap}(\texttt{O}_{\texttt{Main}}, 10),$

$$\begin{aligned}&\texttt{shape}(\texttt{O}_{\texttt{Main}}, \texttt{sprism}), \texttt{shape}(\texttt{O}_{\texttt{Sec}}, \texttt{sprism}),\\ &\texttt{Mu is } 7.05 + 0.57* \simeq(\texttt{distX}(\texttt{O}_{\texttt{Main}}, \texttt{O}_{\texttt{Sec}})) + 0.02* \simeq(\texttt{distY}(\texttt{O}_{\texttt{Main}}, \texttt{O}_{\texttt{Sec}})).\end{aligned}$$

meaning for a tap action, if the two shapes are small prisms, *displX* of O_{Sec} is distributed according to a Gaussian with mean given by Mu, as in Equation 1.

There will be a displacement of a secondary object due to an action on a main object only if the two objects are close enough to interact, otherwise the secondary object will remain in the same position. So, to model that the above Gaussian approximation as in Equation 1 holds only in these cases, we can define the predicate `approx_ok`, which is true when the distance `DX` and `DY` between O_{Main} and O_{Sec} is sufficiently small for the action to have an effect on O_{Sec}. For example, for a *tap*(10) for two small prisms as in Figure 8, we can use the definite clause:

$$\begin{aligned}&\texttt{approx\_ok}(\texttt{tap}, 10, \texttt{sprism}, \texttt{sprism}, \texttt{DX}, \texttt{DY}) \leftarrow \texttt{DX} > 4, \texttt{DX} < 14,\\ &\qquad\qquad\qquad \texttt{DY} > -8, \texttt{DY} < 8.\end{aligned}$$

The smallest x-axis centre-to-centre distance between two objects is $4cm$, so for the objects to interact during a tap their centres need to be between $4cm$ and $14cm$ away on the x-axis. On the y-axis, since the y-axis dimension of a *sprism* is $8cm$, their centres need to be between $-8cm$ and $8cm$ away for an interaction to occur.

Then we just need to add `approx_ok(tap, 10, sprism, sprism, DX, DY)` to the body of the DC clause defining `displX` above. Doing so the displacement $\texttt{displX}(\texttt{O}_{\texttt{Sec}})$ will be defined only when close to $\texttt{O}_{\texttt{Main}}$. Similar rules can be added to enforce the action space.

At this point we can fully model the right arm relational affordances model, as well as the two-arm simultaneous *push*, with the learnt parameters.

For the left arm, given the symmetry of the iCub, the model is equivalent to the model for the right arm mirrored through the plane perpendicular to the table that passes through the centre of the robot. For our model, in the left-arm model all the y-axis values are the same as for the right-arm model, but the x-axis values are the negative of their right-arm equivalent.

In our DC framework, the `displX` and `displY` random variables for left and right arm actions need to be defined by different probability distributions. So, we need to add an extra term to our action atoms to signify the arm performing the action. At this point, we can automatically generate the PPL code for the left arm. For our running example, the equivalent code for Equation 1 for the left arm:

$$\begin{aligned}&\texttt{displX}(\texttt{O}_{\texttt{Sec}}) \sim \texttt{gaussian}(\texttt{Mu}, 0.41) \leftarrow \texttt{tap}(\texttt{O}_{\texttt{Main}}, \texttt{left}, 10), ...\\ &\quad \texttt{Mu is } -7.05 - 0.57* \simeq(\texttt{distX}(\texttt{O}_{\texttt{Main}}, \texttt{O}_{\texttt{Sec}})) - 0.02* \simeq(\texttt{distY}(\texttt{O}_{\texttt{Main}}, \texttt{O}_{\texttt{Sec}})).\end{aligned}$$

At this point we have a relational affordances model for single right and left-arm actions, as well as the simultaneous two-arm actions learnt with the exploration data. Once the program is defined, the inference algorithm based on sampling in [21] or [54] can be used to compute the probability of a query. For example, given our example, one can ask for the probability of the x-axis displacement of the secondary object o_2 being greater than $5cm$ given some initial distances between the main object o_1 and o_2: $P(\simeq(displX(o_2)) > 5|action(o_1, tap, 10), \simeq(distX(o_1, o_2)) = 7, \simeq(distY(o_1, o_2)) = 2)$.

Task (i) is thus achieved, and the robot has a relational affordances model for modelling single-arm left and right-arm actions, and simultaneous (*push*) two-arm actions in the environment.

5. Planning

In this section we present **Task (ii)**, namely **modelling a state transition model** from the relational affordances model, and **Task (iii)**, namely **inferring the next best action** to execute towards reaching the goal. These are required in order for the robot to reach a more complex goal, which requires several actions on objects which can interact with one another during the actions.

5.1. States and Action Representation

This subsection will present **Task (ii)**, learning the state transition model. To model the state transition model of the dynamics of the world we will use the DDCs formalism introduced in Section 2. In our table-top object placement setting, states will have to describe the objects' shape and spatial configurations. An object's position is defined by the x ($coordX$) and y-coordinates ($coordY$) of its centroid. As opposed to [44], where the states were a conjunction of the x and y-axis distance between objects, in order to facilitate planning over multiple steps we will define a state as a conjunction of the $coordX$ and $coordY$ grounded atoms for all the table-top objects: $\wedge_n coordX(o_i, x_i)_t \wedge_n coordY(o_i, y_i)_t$.
The state before executing an action is obtained by the iCub by using its perception to obtain object shapes and centroids. For example, the current state s_t corresponding to Figure 8 (l) before the tap can be seen in Table 3, where o_r is the right (red) object, and o_l the left one.

Table 3: Example states for tap action in Figure 8

s_t	s_{t+1}
$coordX(o_r)_0 : -10.11$	$coordX(o_r)_1 : 0.22$
$coordY(o_r)_0 : 42.70$	$coordY(o_r)_1 : 42.02$
$coordX(o_l)_0 : -3.17$	$coordX(o_l)_1 : 4.26$
$coordY(o_l)_0 : 44.60$	$coordY(o_l)_1 : 43.29$

In our setting we will use high-level goals, represented by a conjunction of spatial relations between objects. For example, the goal can be to place a small prism to the $left$ of a big prism:

$$\mathtt{goal} \leftarrow \mathtt{shape}(\mathtt{O}_1, \mathtt{sprism})_\mathtt{t}, \mathtt{shape}(\mathtt{O}_2, \mathtt{bprism})_\mathtt{t}, \mathtt{left}(\mathtt{O}_1, \mathtt{O}_2)_\mathtt{t}.$$

Note that the goal is true if there exists at least a pair of objects O_1 and O_2 that satisfy *goal*. If we want to achieve the goal for specific objects, we just need to replace the logical variables O_1 and O_2 with the constants of the objects.

Each spatial relation will be defined in terms of the x and y-coordinates of objects. For example:

$$\begin{aligned}
&\texttt{left}(\texttt{O}_1,\texttt{O}_2)_\texttt{t} \leftarrow \simeq(\texttt{coordX}(\texttt{O}_1)_\texttt{t}) > \simeq(\texttt{coordX}(\texttt{O}_2)_\texttt{t}).\\
&\texttt{inshelf}(\texttt{O}_1)_\texttt{t} \leftarrow \simeq(\texttt{coordY}(\texttt{O}_1)_\texttt{t}) > 40.\\
&\texttt{near}(\texttt{O}_1,\texttt{O}_2)_\texttt{t} \leftarrow O_1 \neq O_2, (\simeq(\texttt{coordX}(\texttt{O}_1)_\texttt{t}) - \simeq(\texttt{coordX}(\texttt{O}_2)_\texttt{t}))^2 +\\
&\qquad \simeq(\texttt{coordY}(\texttt{O}_1)_\texttt{t}) - \simeq(\texttt{coordY}(\texttt{O}_2)_\texttt{t}))^2 < 100.
\end{aligned}$$

An object is considered in the shelf ($inshelf$) if the y-axis coordinate greater than $40cm$, corresponding to objects behind the dotted yellow line in Figure 6). Two objects are considered near each other if their centre-to-centre distance is less than $10cm$.

The state transition model can be defined from our learnt affordances model. Using the affordances model, we can compute the object displacements *displX* and *displY* caused by an action with its respective probability distribution. More specifically, the new x and y-axis coordinates of an object defining state s_{t+1} are given by the sum between the old coordinates in state s and the displacement of the object due to the action, or are the same as in the previous state if there is no object displacement. For example, for the x-axis:

$$\begin{aligned}
&\texttt{coordX}(\texttt{O}_1)_{\texttt{t+1}} \sim \texttt{val}(\texttt{X}) \leftarrow \texttt{X is} \simeq(\texttt{coordX}(\texttt{O}_1)_\texttt{t}) + \simeq(\texttt{displX}(\texttt{O}_1)_\texttt{t}).\\
&\texttt{coordX}(\texttt{O}_1)_{\texttt{t+1}} \sim \texttt{val}(\texttt{X}) \leftarrow \texttt{X} = \simeq(\texttt{coordX}(\texttt{O}_1)_\texttt{t}), \texttt{not}(\texttt{displX}(\texttt{O}_1)_\texttt{t}).
\end{aligned}$$

The distribution `val(v)` assigns probability 1 to the value `v`, in this case the value is defined in the body of the clause. While `not(expr)` succeeds if the query `expr` fails or is undefined, in this case when $\texttt{displX}(\texttt{O}_1)_\texttt{t}$ is undefined. In our running example from Figure 8 the next state x and y-axis coordinates due to the $tap(right, 10)$ action can be seen in Table 3. For example, the new x-axis coordinate for the o_r object in the next state s_{t+1} is: $-10.11 + 10.33 = 0.22$.

5.2. Adding Goal Constraints in the Environment

As mentioned in Section 2, the DDC framework allows for the definition of action applicability. The planning algorithm will only select from the applicable actions, i.e., $\texttt{applicable}(\texttt{action})_\texttt{t}$ holds. In a normal household environment there are many objects present, however acting on all of them might not make sense in order to achieve goals (e.g., picking up a fork and a book at the same time), or an action might not be possible in a given object configuration (e.g., an object might be hidden behind another object or out of reach). So we want to define $\texttt{applicable}(\texttt{action})_\texttt{t}$ to enforce any goal constraints in the environment, which will also narrow down the state space search of the robot when doing planning.

We showed in previous work [44] how goal constraints about two-arm actions were added. As a show-case, we show below the precondition that an action should not be executed on a possibly occluded object, that is an object that is behind another object. We define a possibly occluded object, as any object that has a y-coordinate greater than that of another object placed in front of it, and an x-coordinate of its center within $\pm 10cm$ of that object in front of it. This can be defined for example with the help of logical rules as follows:

$$\begin{aligned}
&\texttt{occluded(Obj)}_\texttt{t} \leftarrow \simeq(\texttt{coordY(SecO)}_\texttt{t}) > \simeq(\texttt{coordY(Obj)}_\texttt{t}),\\
&\qquad \simeq(\texttt{coordX(SecO)}_\texttt{t}) > \simeq(\texttt{coordX(Obj)}_\texttt{t}) - \texttt{10},\\
&\qquad \simeq(\texttt{coordX(SecO)}_\texttt{t}) < \simeq(\texttt{coordX(Obj)}_\texttt{t}) + \texttt{10}.\\
&\texttt{applicable(push(X))}_\texttt{t} \leftarrow \texttt{object(X)}, \texttt{not(occluded(X))}.
\end{aligned}$$

Further preconditions for object applicability can be coded depending on the background knowledge one has about the goal the robot needs to execute. For example, some actions might only be desirable on certain objects. The DDC makes it easy to define these goal constraints with the help of logical rules, which in turn limits the search space of the robot when doing planning.

5.3. Proposed Planning Algorithm

This subsection will present **Task (iii)**, by introducing a planning algorithm that will allow us to infer the next best action to execute towards reaching the goal. Once we have defined the DDC state transition model, we adopt a simple sample-based planner[4]. The algorithm starts with an initial state s_0, a goal G and a reward function $r(s, a|G)$ derived from G, which we will define shortly. For each applicable action a_0 it samples N_e episodes following a default policy π_D and averages the obtained rewards to estimate $Q(s_0, a_0)$. Finally, choose the action that maximizes $argmax_{a_0} Q(s_0, a_0)$.

Algorithm 1 Planning algorithm for finding the best action towards the goal

```
1: procedure FindBestAction($s_0$)
2:     for $a_0 \in$ {applicable actions in $s_0$} do
3:         sample $N_e$ episodes of length $d$ from $(s_0, a_0)$ with default policy $\pi_D$
4:         $Q(s_0, a_0) \leftarrow \frac{\sum_{t=0}^{d} r_t}{N_e}$
5:     end for
6:     return $argmax_{a_0} Q(s_0, a_0)$
7: end procedure
```

The default policy π_D is a uniform distribution in the applicable actions in the state s: $\pi_D(a|s) = uniform(\{\text{applicable actions in } s\})$. In DDC this becomes:

[4]An improved, but more involved planner for DDCs, is described in [51], but is outside the scope of this paper.

$\mathtt{policy_t} \sim \mathtt{uniform(List)} \leftarrow \mathtt{findall(A, applicable(A)_t, List)}.$

This algorithm is naive because it evaluates the Q function using a flat policy, indeed it does not have a policy improvement mechanism. In a discrete state space policy improvement would be easy to implement, however in a hybrid relational domain this requires a non-trivial representation to store the Q function.

Despite its simplicity, a similar strategy has been used in Monte Carlo Tree Search with the name "Flat Monte Carlo" [5] with good results in computer games. In addition, this algorithm is faster than sparse sampling as showed in preliminary experiments.

In our setting we want to reach a goal G in the minimum number of steps. That is minimizing the expected cost of reaching the goal assuming each action has the same cost c in any state. This is equivalent to maximising the expected reward in an MDP with a reward function $r(s, a|G) = 0$ for $s \models G$, and $r(s, a|G) = -c$ otherwise, with $c > 0$. The value assigned to c is not important (e.g. $c = 1$), since it gives the same policy. In addition, we assume a maximum number of steps d to reach the goal (finite horizon MDP). While the state transition model $p(s_{t+1}|s_t, a_t)$ is defined as a set of DDCs of the form $\mathtt{h_{t+1}} \sim \mathcal{D} \leftarrow \mathtt{body_t}$. $\mathtt{h_{t+1}}$ defines a random variable in the next state s_{t+1} with distribution $\mathcal{D}$ when $\mathtt{body_t}$ holds. The body of the clause includes the action and other conditions that specify when the clause applies.

To implement the planner we use DCPF [52]: a particle filter for DDCs. We initialize the particles with the initial state s_0 (that contains the last observed object positions) and generate episodes with the default policy as described in the algorithm. We assume full observability, thus the observation model is not needed (as in HMM/DBN). Therefore, we exploit DCPF only to generate sequential samples and store the total reward without observations. The generation of sequential samples is similar to the procedure described in Section 2.2 for DCs.

As an example, suppose in our model, we observe the initial state with two *sprism* objects, object $\mathtt{o_1}$ at coordinate $(10, 35)$ and object $\mathtt{o_2}$ at coordinate $(12, 55)$. The goal is to have the two objects near each other, according to the previously defined `near` relation (centre-to-centre distance of less than $10cm$). We limit the sampled episode length to $d = 4$. Table 4 shows the output Q function estimated by running the planning algorithm for each of the applicable initial actions. Therefore, the best first action to execute is: $\mathtt{push(o_1, left, 25)}$.

6. Evaluation and Results

In this section, we shall report on an empirical evaluation of our method using the iCub robotics platform. First, we will summarize our previous results in Section 6.1 where we evaluate the differences between the relational affordances model and the multiple Bayesian Network models needed to cope with multiple objects in a model using discrete variables only. This comparison aims at evaluating the most important property of the model, the action-object selection

Table 4: Q function estimate obtained by running planning algorithm

Action	Q function estimate
$\texttt{action}(\texttt{push}(\texttt{o}_1, \texttt{left}, 15))$	-3.718
$\texttt{action}(\texttt{tap}(\texttt{o}_1, \texttt{left}, 10))$	-4.434
$\boldsymbol{action(push(o_1, left, 25))}$	$\mathbf{-2.402}$
$\texttt{action}(\texttt{tap}(\texttt{o}_1, \texttt{left}, 20))$	-4.918
$\texttt{action}(\texttt{tap}(\texttt{o}_1, \texttt{right}, 10))$	-4.224
$\texttt{action}(\texttt{tap}(\texttt{o}_1, \texttt{right}, 20))$	-4.999

for an expected effect. Secondly, in section 6.2, we shall investigate whether the learned relational affordance models can be used for table-top object manipulation in a setting where a sequence of actions is necessary to reach a goal. Section 6.2 significantly extends our approaches to relational affordances in that more complex multi-action plans are needed to reach the goal in a mixed discrete continuous setting. The evaluation is based on the goal success, which is the number of times that the requested goal is accomplished, a very challenging task given the uncertainty coming from the model and the robot skills.

6.1. Comparison to Affordances Modelled with BNs

Previous research [46] has also investigated how our relational affordances approach compares with affordances modelled with the aid of BNs. We will present here an overview of those results.

The experiments consisted of 200 settings in simulation with an iCub, each setting with six objects of two possible shapes (*square* and *rect*, as seen from above). Three objects are always in the field of action of the robot, though the robot might not be able to perform all the actions on every object as this might violate some rules (e.g. action space for that action, interference with the hand from nearby objects). The other three objects are placed behind these and might interact with them when performing an action. All the objects are randomly placed within certain margins and have a random shape. One such placement is shown in Figure 10(1). In this setting, we execute all possible actions with the iCub to get real-world matching effects. Given these effects and the object properties, the action was predicted and compared against the ground truth action performed.

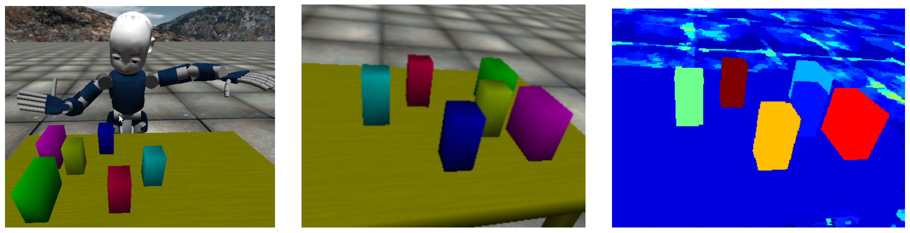


Figure 10: Six objects(1), left eye image(2) and its segmented objects(3).

In these experiments, the discrete-only variables version [5] of the affordances model was used, composed of the following variables: (i) Object shape, (ii) relative location (2D vector) between objects' centroids, which corresponds to the relative distance and orientation angle variables. In the continuous model for planning experiments (shown in Fig. 9, the equivalent variables are displX_ and displY_ After an initial domain analysis, they were discretized as follows: the relative distance in 4 clusters separated at $6cm$, $10cm$ and $14cm$; the orientation angle in 8 clusters of $45\,^\circ$. The action was one of the three possible actions: *grasp*, *push*, *tap*. The effects are the displacement and its angle orientation, which are similarly clustered, with the displacement clusters separated at $1cm$, $3cm$ and $5cm$, and contact between pairs of objects (possible values: $true/false$) [46].

The relational affordances model predicts the action and main object by calculating $\arg\max_{A,ObjMain} P(A_{ObjMain}|O,E)$ and comparing to the real action-object pair. The best predicted object to act on is computed by summing over all possible actions in the above formula, and the best predicted action by summing over all possible main objects. The six-object BN model has an action node with 9 possible values (3 actions for each of the 3 reachable objects), and the MAP estimate is computed to find out the predicted object-action duo. For the BN model, the data was split into two sets, where one was used for training and one for testing. For each BN, the results shown in Table 5 are averaged over 6 runs.

Table 5: Action prediction in six-object scenarios [46].

	Prediction Task	Total exp.	Success	Pct.
ProbLog Model	$ObjMain$	200	149	74.5%
	A	200	137	68.5%
	$ObjMain$ and A	200	116	58%
BN Model (avg over 6 train/test sets)	$ObjMain$	100	67	67%
	A	100	69.2	69.2%
	$ObjMain$ and A	100	67	67%

The two approaches have comparable results; the PPL model is slightly better at predicting the object to act on, while the BN is slightly better for the object-action duo. However, the PPL model can be used without being changed for any number of objects, while the BN needs to be learned again and its size is approximately proportional to the number of objects in the environment (a six-object setting BN already has 65 nodes). In addition, the PPL enables transferring structural parts of the model (e.g. abstract action-effect rules) to similar domains with more, less or other types of objects. If we want improved

[5] The hybrid model was developed later

accuracy tailored to a setting, the PPL model can also be trained with more data [46].

We also looked at some learning statistics of the PPL model (summarized in Table 6): i) the confidence of the predictions (i.e. the value of $P(A|O,E)$) and (ii) the number of correct effects produced by an incorrect predicted action.

Table 6: Learning statistics of the PPL affordances model [46].

Prediction Result	Statistic	Pct.
Success	$ObjMain$ confidence	73.1%
	A confidence	90.1%
	$ObjMain$ and A confidence	62.6%
Failure	Correct effects	67.2%

We see that the confidence of predicting an action is very high, while (as expected) that of the object-action duo is lower. When our prediction is wrong, executing the action still manages to produce about $\frac{2}{3}$ of the 27 effects correctly, sometimes a good compromise in complex scenarios [46].

The affordances model is able to achieve the expected configuration of the objects in a very complex setting, where the number of objects to act upon is between 3 and 6. The model is able to cope with the object selection, even though the model does not enforce it, because the goal is to reach a configuration of the objects independently of which object is acted upon. On one hand, the BN models perform better for some tasks but at the cost of having multiple models running at the same time, which includes learning multiple models for each scenario and selecting the model at runtime. On the other hand, our PPL model is able to cope with multiple objects, being able to select the action and object on scenarios with multiple objects (3-6), with the additional advantage of learning just one model with interactions between two objects.

6.2. Continuous Model performance in two-arm manipulation

In these experiments we evaluate the improvement provided by the continous model LCGBN, performing action selection as in the experiments of the previous section. These experiments were presented previously [44], where the PR2 robot in simulation[6] is performing a single arm actions and two-arm actions using the continuous relational affordance model. Similar to the experiments in Section 6.1, we investigated the action prediction on a scenario with four or six objects on the table. The relational affordance model was learnt from 300 exploratory single-arm actions, which were initially modelled by the same LCGBN as in this paper in Figure 9. The model was then extended by symmetry to the other arm, and extended by the use of logical rules to represent constraints on two-arm robot actions (e.g., one arm acts on an object, while the other acts on an

[6]Gazebo simulator

object that interacts with the first object). We then extended the model by modelling either the sequential or simultaneous use of the arms.

We ran this experiment on 100 action prediction scenarios, where given the set of object properties O and effects E, we tried to predict the best left and right arm actions A_L, and A_R, and the best objects the two arms act on, O_L and O_R respectively. We use the SRL model to compute

$$\arg\max_{A_L,O_L,A_R,O_R} P(twoArmA(A_L,O_L,A_R,O_R)|O,E)$$

and compare these to the ground truth action-object pairs.

Table 7: Two-arm model action prediction results [44]

	Total exp.	Success	Pct.
Correct two-arm object(s)/actions	100	68	68%
Correct manipulated objects	100	74	74%
Correct left/right actions	100	68	**68 %**
Random choice			2.78%
Random choice given constraints			9.52%

Table 7 illustrate that the single arm actions (left/right) modeled by the mix of continuous and discrete variables perform better than the discrete model. below. For comparison, the robot picks the correct two-arm actions and object(s) to act on in 68% of the cases in this continuous two-arm experiment, compared to the single-action affordance model which was 58% (See Table 5). This shows the improvement of around 10% on single-arm action selection by using the LCGBN model, and also the viability of a two-arm model for action prediction.

6.3. Sequential planning based on the relational affordances model

We now investigate whether our learnt two-arms probabilistic relational affordances model can be used successfully in a table-top object manipulation setting, where a sequence of actions of the two arms is necessary to reach a goal. The goal success of the planning is influenced mainly by: (i) uncertainty of the affordances model, (ii) uncertainty of the motor capabilities of the robot and (iii) uncertainty of the visual skills. Thus, the planning algorithm should be able to cope with accumulated errors from these factors, a very challenging task.

For the experiments, we use the iCub in a real setting. The iCub uses its perceptual capabilities to detect the shapes and positions of the objects on the table. After detecting the objects in the scene, the observations are used by the planner, which infers the best action to execute towards the goal. The iCub executes this action. If the goal has not been reached, this process is continued. We limit in the planner the number of iCub actions to a sequence of maximum 4 actions. Figure 11 shows the data flow between the main software components, denoted as boxes.

Several software libraries and modules available at the iCub software repository were utilized and/or extended in order to learn the model and perform the experiments. The main software modules include the parametric actions module, the stereo vision, the image segmentation and the planner. Finally, there is a master module that controls the perception-planner-action loop, which runs while the goal has not been reached and the number of actions needed to reach the goal is less than four.

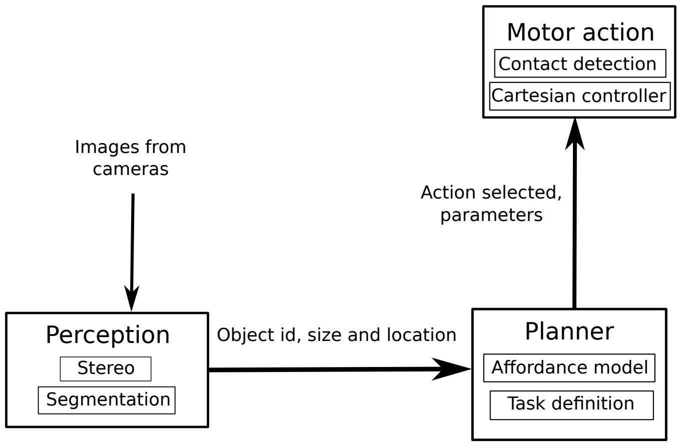


Figure 11: Main software components running for the experiments

The parametric actions module is largely inspired on the actionsRenderingEngine [7] but defining an additional action, the two-hands push. In addition, all the actions are parametrized with the arm selection and hand displacement vector, which follows the description of Section 4.1. Similar to the actionsRenderingEngine, our parametric actions module is built on top of the actionsPrimitive library [8] that allows to define actions as a set of end-effector points in the Cartesian space. The actionsPrimitive library provides interfaces to stop the execution if the forces on the arms exceed a threshold, which we utilize to stop the execution of the pre-action and post-action motions in case of contacts with the table or other objects. The stereo vision module [9] utilizes the OpenCV [10] stereo algorithm in [23], considering the kinematic chain of the iCub and the mechanical mounting error of its eyes. After an initial calibration, the stereo module provides: stereo triangulation and pixel-to-3D mappings. The image

[7]https://github.com/robotology/icub-main/tree/master/src/modules/actionsRenderingEngine
[8]https://github.com/robotology/icub-main/tree/master/src/libraries/actionPrimitives
[9]https://github.com/robotology/stereo-vision
[10]http://opencv.org/

segmentation module [11] provides a YARP [42] wrapper to the confidence-based edge detector and mean shift segmentation algorithm [12]. The planner module is a script written in the DDCs [13] that receives the input from the perception module in the form of clauses and outputs the action to be performed by the robot for a specific goal, so each goal has an associated script. The master module is a set of Python [14] classes and a script that integrates the perception, planner and parametric actions in a loop that aims to reach the goal while the current action leads to the goal in four steps.

Each experiment is set up as follows. The iCub is placed in front of a table with several objects on it in front of the robot, and the perception-planner-action loop is executed until the success or failure of the high-level goal. The robot is also given a high-level goal it needs to achieve. One such example of initial setting and given goal following the definitions in Sec. 4.1 is:

$$\begin{array}{ll}
\mathit{Initial}: & \mathit{shape}(o_1, \mathit{sprism}), \mathit{shape}(o_2, \mathit{bprism}), \mathit{shape}(o_3, \mathit{bprism}) \\
& \mathit{coordX}(o_1, -22.3), \mathit{coordY}(o_1, 37.5), \\
& \mathit{coordX}(o_2, -10.5), \mathit{coordY}(o_2, 43.2), \\
& \mathit{coordX}(o_3, 9.6), \mathit{coordY}(o_3, 41.5), \\
\mathit{Goal}: & \mathit{shape}(Q, \mathit{sprism}), \mathit{shape}(R, \mathit{bprism}), \mathit{near}(Q, R)
\end{array}$$

This initial configuration for the iCub can be seen in Figure 12(l). The final configuration shows the goal being reached successfully by placing the red object next to the orange object, where next means the distance between object centroids less than a predefined value (10cm).

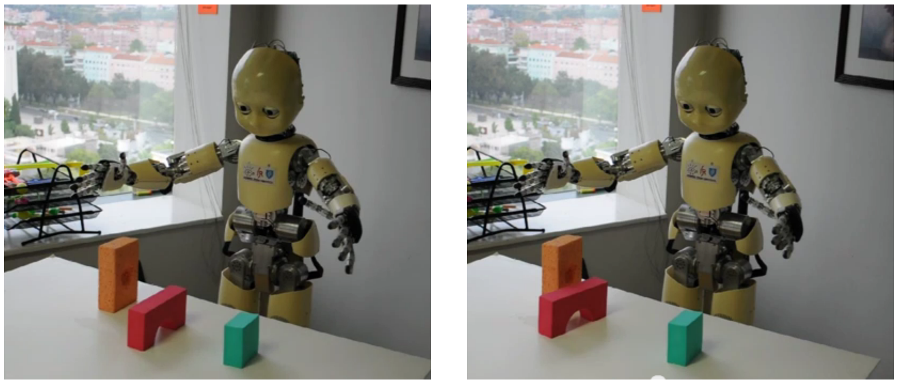


Figure 12: Initial object placement (l), and final object locations (r). The goal is two of the objects in the environment close to each other

For the goals we assign to the robot, we will use an object placement game

[11] https://svn.code.sf.net/p/robotcub/code/trunk/iCub/contrib/src/poeticon/poeticonpp/edisonSegmentationModule/
[12] http://coewww.rutgers.edu/riul/research/code/EDISON/
[13] https://dtai.cs.kuleuven.be/ml/systems/DC
[14] https://www.python.org/

setting, inspired from [38]. The high-level goal presented to the robot will be one of the following four, involving spatial relations between objects, in increasing order of complexity:

- Goal 1. Place any two objects near each other
- Goal 2. Place all *sprism* to the left of all *bprism*
- Goal 3. Place two objects in the shelf, namely: a *sprism* and a *bprism*. The *sprism* must be to the left of a *bprism*
- Goal 4. Place all objects in the shelf, all *sprism* to the left of all *bprism*

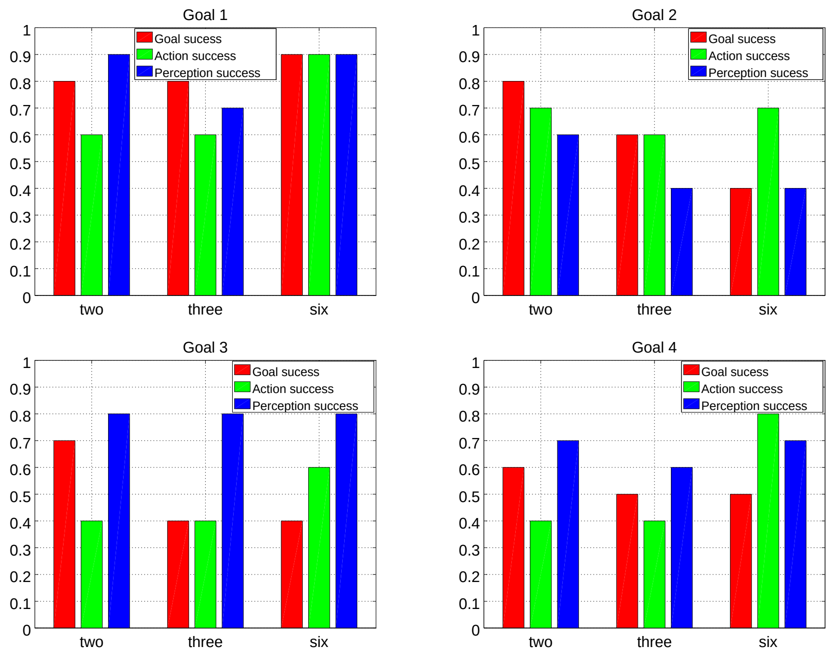


Figure 13: Performance over goals and discriminated by number of objects. Goal 1 is to place two of the objects in the world close to each other. Goal 2 is to place all the small objects on the left of all middle size ones. Goal 3 is to place two objects in shelf, a small and a middle ones, being the small on the left side of the middle one. Goal 4 is to place all the objects in shelf, placing all small on the left of all middle ones

For each type of goal, a person placed the objects on the table for the initial table top setting. Every goal was repeated 30 times, splitted into 10 configurations with two objects, 10 configurations with three objects and 10 configurations with six objects. The tests with six objects were performed in simulation. Thus, 120 experiments were performed in total, where 80 were performed with the real robot. It is important to remark that the affordances model learning was done in simulation, meaning that the uncertainties coming from the friction, illumination changes and the arm and head motion execution were not included in the affordances model. These variations in perception, action and physical properties of the objects decrease the success criteria, but will allow to demonstrate experimentally the robustness of the affordances model to "unexpected" perception and action. The success of achieving goals is analysed by the following criteria:

- Goal Success. The ratio between the number of reached goals and the number of attempts.
- Motor Skills Success. For each goal attempt, this criterion is 1 if every action performed by the robot led to effects observed during the exploratory phase. If any of the effects were not observed in the previous exploratory phase, this criterion is 0.
- Perceptual Skills Success. For each goal attempt, this criterion becomes 1 if all the object properties observed (object centroid and object size) during the sequence of actions were computed in a range of admissible values (+/- 2 cm). Any object property out of the admissible range will lead to 0.

It is worth stressing that since the data for learning was acquired in the simulation, there will be unobserved/unpredicted effects when the robot operates on a real environment. Examples of such unobserved effects not sampled during the exploratory phase in simulation include: (i) Errors in the motion controllers that turned into trajectories that collide with the table and objects, which in simulation do not happen; (ii) actions that cause large rotations of the objects (more than 30°), which in simulation are usually smaller due to the friction model.

The "Motor Skill Success" along with the "Goal Success" evaluate the robustness of the decision making process to uncertainty in the action execution, while the "Perceptual Skills Success" along with the "Goal Success" evaluate the robustness of the decision making to uncertainty in the perception.

The results summarized in Table 8 were gathered from 120 experiments and allow us to draw a number of observations: Firstly, the "Goal Success" is larger than the "Motor Skills Success" for the two-object and three-object subsets, so the perception + planner modules are robust enough to deal with failures in the action execution. However, the six-object experiments do not follow the same pattern because of three types of situations: (i) The experiments were performed in simulation so the motor and perceptual uncertainty are less than in the real world and (ii) The planner failed to find a solution in 4 steps a lot more times than in the other cases. Secondly, the "Goal Success" is below the "Perceptual Skills Success" as expected, because of the propagation of uncertainty from the perceptual module to the planner. The execution of the actions lead to unreachable states due to noisy observations. Thirdly, we observe that in a "Perfect world" where neither the action or perception errors occur, the affordances model + planner is able to reach the goals as expected. Finally, we remark in the last row of Table 8 the success criteria for every action

Table 8: Consolidated results by type of goal. Every goal has three diferent setups (2,3 and 6 objects). The two-objects row averages the two-row settings over all the goals and so on. The main evaluation criterion of the relational affordances model + planning algorithm is the Goal Success, which evaluates the robustness of the decision making to uncertainty in the action execution and perceptual capabilities. The motor skills success and perceptual skills success acts as reference for the goal success.

	Goal success(%) (± std. dev.)	Average # of actions (± std. dev.)	Motor skills success (%) (± std. dev.)	Perceptual skills success(%) (± std. dev.)
Two objects	**72.5**(9.57)	2.675(0.478)	52.5(15)	75(12.9)
Three objects	**57.5** (17.07)	3.725(1.07)	50(11.54)	62.5(17.07)
Six objects	**55** (23.8)	1.9(0.82)	75(12.9)	70(21.6)
All goals	**61.66** (18.07)	2.76(1.08)	59.16(16.76)	69.16(16.76)
Perfect world	**95.23**	-	100	100
Per action exec.	-	-	85.54	83.73

executed by the robot, which correspond to the highest possible performances of the affordances model + planner. Remark that the total performance (62%) is good when compared to the highest possible performance (85%). This indicates that the affordances model is very robust to the unobserved/unpredicted effects, and the generalization properties of the model.

Figure 13 shows the performance measures marginalized by task and complexity, where it can be observed that all of the success criteria decrease their values as there are more objects in the world. This is expected, since the affordances model was learned in an simplified version of the world, where only two objects were visible to the robot. Nevertheless, the affordances model + planner is able to reach goals for table top configurations that were not seen before by the robot, even in test scenarios where the number of objects is three times more than the training scenario. In order to increase the success rate of the Goals 3 and 4, the addition of motion planning capabilities for action execution will avoid undesired contacts with other objects.

The results presented in this section show that basic motor and perceptual skills (i.e. low-level visuomotor features) can be integrated with highly abstract representations of the world (i.e. objects and their interactions), reaching successfully abstract goals in a multi-object table top environment. The integration between the low-level and high-level skills consider a hybrid representation of the world, where continuous and discrete variables are mixed in the learning and inference processes. In addition, the planning capabilities added to the affordances model allow to solve for several object arrangements that have not been taught to the robot previously.

Experiments were run on computers with Intel Core $i5-2500$ $3.3GHz$ processors, $6MB$ cache, and $8GB$ memory. We implemented our model with DDCs and used 1000 samples for planning. Each planning step took about $15(\pm5.2)$ secs. for two object scenarios, $40(\pm11.1)$ secs. for three object scenarios, and $220(\pm22.5)$ secs. for six object scenarios. The action execution on the real iCub tool $0.7(\pm0.12)$ seconds.

7. Conclusion and Future Work

We have presented a relational affordances model that is able to deal with continuous and discrete variables, being able to select correctly the action from a large repertoire while reaching for abstract goals. The robot first learns an affordances model through its interactions with the environment and is then able to use the model to plan for and realize its goals. The relational model generalizes across any number of objects, and can be applied in complex sorting tasks that need planning of manipulation actions in order to reach a goal configuration. The generality of the model is shown by training the model only with two-object interactions, which provide the necessary information to perform decision making (i.e. action selection) in scenarios having up to six objects. Configurations in a table-top setting were used, and where acting on one object may produce also effects on neighbouring objects. The experiments showed that the robot is able to realize fairly complex goals while only having a set of simple perceptual capabilities. The goals are expressed in logic programming DC syntax, which has the advantage of associating a semantic goal description to a set of fairly complex rules. The planning algorithm is rather simple, but is able to solve a larege number of tasks where the number of objects in the world is not known a priori. The main limitation of the planning algorithm is the computational complexity, which is exponential in the number of objects in the scenario. When compared to similar works on affordances, we provide a comprehensive affordance model that (i) Considers the relations between mulitple objects, (ii) model jointly continuous and discrete variables, and (iii) integrate planning algorithms in order to solve sequential decision making problems in relational scenarios. The model is not only relevant to affordances, but also to show the promise of a probabilistic programming approach for robotics.

There are many opportunities for further work. One question is whether it is possible to learn even more complex affordances that would take into account tactile perception during the exploration phase. Another one is whether the addition of spatial relations from conventional motion planning and the use of other background rules could improve the accuracy of the learned models and the performance of the planner. Finally, we are currently also working on ways to directly learn the distributional clauses from observations, rather than indirectly through the learning via the generalization of a learned Bayesian network. The addition of those improvements to the relational affordances model and the planning algorithm should allow to solve more complex tasks and to improve the performance of the robot.

Acknowledgements

This work was partly supported by the EU FP7 Project FIRST-MM, the IWT via scholarships for BM and DN, the Research Foundation Flanders, and the KULeuven BOF. Also by Research Infrastructure 22084-01/SAICT/2016 - Robotics, Brain and Cognition (RBCog-Lab) and Project FCT UID/EEA/50009/2013

Bibliography

[1] Amant, R. S., 1999. Planning and user interface affordances. In: Proceedings of the 1999 international conference on Intelligent user interfaces. pp. 135–142.

[2] Berenson, D., Srinivasa, S. S., 2008. Grasp synthesis in cluttered environments for dexterous hands. In: Humanoids.

[3] Blum, A. L., Langford, J. C., 1998. Probabilistic planning in the graphplan framework.

[4] Brown, S., Sammut, C., 2013. A relational approach to tool-use learning in robots. In: Inductive Logic Programming.

[5] Browne, C., Powley, E. J., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S., 2012. A survey of monte carlo tree search methods. IEEE Trans. Comput. Intellig. and AI in Games 4 (1), 1–43.

[6] Cakmak, M., Dogar, M. R., Ugur, E., Sahin, E., 2007. Affordances as a framework for robot control. In: International Conference on Epigenetic Robotics (EpiRob).

[7] Calinon, S., Guenter, F., Billard, A., 2007. On learning, representing, and generalizing a task in a humanoid robot. Systems, Man, and Cybernetics, Part B: Cybernetics 37 (2), 286–298.

[8] Christoudias, C. M., Georgescu, B., Meer, P., Georgescu, C. M., 2002. Synergism in low level vision. In: ICPR.

[9] Cosgun, A., Hermans, T., Emeli, V., Stilman, M., 2011. Push planning for object placement on cluttered table surfaces. In: IROS.

[10] Şahin, E., Çakmak, M., Doğar, M. R., Uğur, E., Üçoluk, G., 2007. To afford or not to afford: A new formalization of affordances towards affordance based robot control. Adaptive Behavior 15 (4), 447–472.

[11] De Raedt, L., 2008. Logical and Relational Learning. Springer.

[12] De Raedt, L., Kersting, K., 2008. Probabilistic inductive logic programming. In: Probabilistic Inductive Logic Programming. pp. 1–27.

[13] De Raedt, L., Kimmig, A., Toivonen, H., 2007. Problog: A probabilistic Prolog and its application in link discovery. In: IJCAI.

[14] Emeli, V., Kemp, C., Stilman, M., 2011. Push planning for object placement in clutter using the PR-2. In: IROS PR2 Workshop.

[15] Fikes, R. E., Nilsson, N., 1971. STRIPS: A new approach to the application theorem proving to problem solving. Artificial Intelligence 5 (2), 189–208.

[16] Fritz, G., Paletta, L., Breithaupt, R., Rome, E., Dorffner, G., 2006. Learning predictive features in affordance based robotic perception systems. In: IROS. pp. 3642–3647.

[17] Geib, C., Mourão, K., Petrick, R. P. A., Pugeault, N., Steedman, M., Krueger, N., Wörgötter, F., 2006. Object action complexes as an interface for planning and robot control. In: Workshop: Towards Cognitive Humanoid Robots at IEEE RAS.

[18] Getoor, L., Taskar, B., 2007. Introduction to statistical relational learning. The MIT press.

[19] Gibson, J. J., 1979. The Ecologial Approach to visual perception. Boston: Houghton Mifflin.

[20] Gienger, M., Toussaint, M., Goerick, C., 2008. Task maps in humanoid robot manipulation. In: IROS.

[21] Gutmann, B., Thon, I., Kimmig, A., Bruynooghe, M., Raedt, L. D., 2011. The magic of logical inference in probabilistic programming. CoRR abs/1107.5152.

[22] Hertzberg, J., Chatila, R., 2008. AI reasoning methods for robotics. In: Handbook of Robotics. Springer, pp. 207–223.

[23] Hirschmuller, H., Feb 2008. Stereo processing by semiglobal matching and mutual information. Pattern Analysis and Machine Intelligence, IEEE Transactions on 30 (2), 328–341.

[24] Jain, D., Mösenlechner, L., Beetz, M., 2009. Equipping robot control programs with first-order probabilistic reasoning capabilities. In: ICRA. pp. 3626–3631.

[25] Jetchev, N., Toussaint, M., 2010. Trajectory prediction in cluttered voxel environments. In: ICRA.

[26] Kearns, M., Mansour, Y., Ng, A. Y., 2002. A sparse sampling algorithm for near-optimal planning in large markov decision processes. Mach. Learn. 49 (2-3), 193–208.

[27] Kjærulff, U. B., Madsen, A. L., 2005. Probabilistic networks - an introduction to Bayesian networks and influence diagrams. Aalborg University.

[28] Kjellström, H., Romero, J., Kragić, D., 2011. Visual object-action recognition: Inferring object affordances from human demonstration. Computer Vision and Image Understanding 115 (1), 81–90.

[29] Kocsis, L., Szepesvári, C., 2006. Bandit based Monte-Carlo planning. In: ECML. pp. 282–293.

[30] Kragic, D., Björkman, M., Christensen, H. I., Eklundh, J. O., 2005. Vision for robotic object manipulation in domestic settings. Robotics and Autonomous Systems 52 (1), 85–100.

[31] Krüger, N., Geib, C., Piater, J., Petrick, R., Steedman, M., Wörgötter, F., Ude, A., Asfour, T., Kraft, D., Omrčen, D., Agostini, A., Dillmann, R., 2011. ObjectAction complexes: Grounded abstractions of sensorymotor processes. Robotics and Autonomous Systems 59 (10), 740–757.

[32] Kruger, N., Piater, J., Worgotter, F., Geib, C., Petrick, R., Steedman, M., Asfour, T., Kraft, D., Hommel, B., Agostini, A., Kragic, D., Eklundh, J.-O., Kruger, V., Torras, C., Dillmann, R., 2009. A formal definition of object-action complexes and examples at different levels of the processing hierarchy. Computer and Information Science, 1–39QC 20120426.

[33] Krunic, V., Salvi, G., Bernardino, A., Montesano, L., Santos-Victor, J., 2009. Affordance based word-to-meaning association. In: ICRA.

[34] Kushmerick, N., Hanks, S., Weld, D. S., 1995. An algorithm for probabilistic planning. Artificial Intelligence 76 (1-2), 239–286.

[35] Lang, T., Toussaint, M., 2009. Approximate inference for planning in stochastic relational worlds. In: ICML. pp. 585–592.

[36] Lang, T., Toussaint, M., 2010. Planning with noisy probabilistic relational rules. Journal of Artificial Intelligence Research 39, 1–49.

[37] Lang, T., Toussaint, M., 2010. Planning with noisy probabilistic relational rules. Journal of Artificial Intelligence Research (JAIR) 39, 1–49.

[38] Lopes, M., Melo, F. S., Montesano, L., 2007. Affordance-based imitation learning in robots. In: IROS. pp. 1015–1021.

[39] Lorken, C., Hertzberg, J., 2008. Grounding planning operators by affordances. In: International Conference on Cognitive Systems.

[40] Lungarella, M., Metta, G., Pfeifer, R., Sandini, G., 2003. Developmental robotics: A survey. Connection Science 15, 151–190.

[41] Mcdermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D., 1998. PDDL - The Planning Domain Definition Language. Tech. rep., CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

[42] Metta, G., Fitzpatrick, P., Natale, L., 2006. YARP: Yet another robot platform. International Journal on Advanced Robotics Systems 3 (1), 43–38.

[43] Metta, G., Sandini, G., Vernon, D., Natale, L., Nori, F., 2008. The iCub humanoid robot: an open platform for research in embodied cognition. In: PerMIS.

[44] Moldovan, B., De Raedt, L., 2014. Learning relational affordance models for two-arm robots. In: IROS.

[45] Moldovan, B., Moreno, P., van Otterlo, M., 2013. On the use of probabilistic relational affordance models for sequential manipulation tasks in robotics. In: ICRA.

[46] Moldovan, B., Moreno, P., van Otterlo, M., Santos-Victor, J., De Raedt, L., 2012. Learning relational affordance models for robots in multi-object manipulation tasks. In: ICRA.

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

[48] Mourão, K., Petrick, R. P. A., Steedman, M., 2010. Learning action effects in partially observable domains. In: ECAI. pp. 973–974.

[49] Murphy, K., et al., 2001. The Bayes net toolbox for Matlab. Computing science and statistics 33 (2), 1024–1034.

[50] Murphy, K. P., 2002. Dynamic bayesian networks: Representation, inference and learning. Ph.D. thesis, University of California, Berkeley.

[51] Nitti, D., Belle, V., De Raedt, L., 2015. Planning in discrete and continuous Markov decision processes by probabilistic programming. In: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD) 2015, Part II. Vol. 9285 of Lecture Notes in Computer Science. Springer International Publishing, pp. 327–342.

[52] Nitti, D., De Laet, T., De Raedt, L., 2014. Relational object tracking and learning. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2014. pp. 935–942.

[53] Nitti, D., De Laet, T., De Raedt, L., 2016. Probabilistic logic programming for hybrid relational domains. Machine Learning, 1–43.

[54] Nitti, D., Laet, T. D., Raedt, L. D., 2013. A particle filter for hybrid relational domains. In: Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 2764–2771.

[55] Pasula, H. M., Zettlemoyer, L. S., Kaebling, L. P., 2004. Learning probabilistic relational planning rules. In: ICAPS. pp. 73–82.

[56] Pattacini, U., Nori, F., Natale, L., Metta, G., Sandini, G., 2010. An experimental evaluation of a novel minimum-jerk cartesian controller for humanoid robots. In: IROS.

[57] Sato, T., 1995. A statistical learning method for logic programs with distribution semantics. In: ICLP. pp. 715–729.

[58] Sinapov, J., Stoytchev, A., 2007. Learning and generalization of behavior-grounded tool affordances. In: ICDL. pp. 19–24.

[59] Steedman, M., 2002. Formalizing affordance. In: Annual Meeting of the Cognitive Science Society. pp. 834–839.

[60] Stoytchev, A., 2005. Behavior-grounded representation of tool affordances. In: ICRA. pp. 3060–3065.

[61] Stulp, F., Beetz, M., 2008. Combining declarative, procedural, and predictive knowledge to generate, execute, optimize robot plans. Robotics and Autonomous Systems 56, 967–979.

[62] Sutton, R. S., Barto, A. G., 1998. Reinforcement Learning: An Introduction. MIT Press.

[63] Thrun, S., Burgard, W., Fox, D., 2005. Probabilistic Robotics. MIT Press.

[64] Toussaint, M., Plath, N., Lang, T., Jetchev, N., 2010. Integrated motor control, planning, grasping and high-level reasoning in a blocks world using probabilistic inference. In: ICRA.

[65] Ugur, E., Dogar, M. R., Cakmak, M., Sahin, E., 2007. The learning and use of traversability affordance using range images on a mobile robot. In: ICRA. pp. 1721–1726.

[66] Ugur, E., Sahin, E., Oztop, E., 2009. Affordance learning from range data for multi-step planning. In: International Conference on Epigenetic Robotics (EpiRob).

[67] Vahrenkamp, N., Berenson, D., Asfour, T., Kuffner, J., Dillmann, R., 2009. Humanoid motion planning for dual-arm manipulation and re-grasping tasks. In: IROS. pp. 2464–2470.

[68] van Otterlo, M., 2009. The Logic of Adaptive Behavior. IOS Press, Amsterdam, The Netherlands.

[69] Wiering, M. A., van Otterlo, M., 2012. Reinforcement Learning: State-of-the-Art. Springer.

[70] Wörgötter, F., Agostini, A., Krüger, N., Shylo, N., Porr, B., 2009. Cognitive agents – a procedural perspective relying on the predictability of object-action complexes (OACs). Robotics and Autonomous Systems 57, 420–432.

[71] Younes, H. L., Littman, M. L., 2004. Ppddl1. 0: An extension to pddl for expressing planning domains with probabilistic effects.

[72] Zamani, Z., Sanner, S., Fang, C., 2012. Symbolic dynamic programming for continuous state and action mdps. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence.

[73] Zettlemoyer, L. S., Pasula, H., Kaelbling, L. P., 2005. Learning planning rules in noisy stochastic worlds. In: AAAI. pp. 911–918.

[74] Zöllner, R., Asfour, T., Dillmann, R., 2004. Programming by demonstration: dual-arm manipulation tasks for humanoid robots. In: IROS. pp. 479–484.