Learning the Consequences of Actions: Representing Effects as Feature Changes

Mathias Rudolph, Manuel Mühlig, Michael Gienger, Hans-Joachim Böhme

2010

Learning the Consequences of Actions: Representing Effects as Feature Changes

Mathias Rudolph   Manuel Mühlig   Michael Gienger   Hans-Joachim Böhme

Abstract— **In advanced Programming by Demonstration (PbD) it is important to give a robot the ability to understand the effects of an action. This ability can enable a robot to not only *mimic* an action but to *imitate*, by determining whether an action succeeded or not, or to *emulate*, by finding another action that causes the same effects as observed. In this paper we propose a system that uses a *Bayesian Network* structure to store actions as a representation of their effects. The effects in turn are implicitly stored as representation of feature changes in the perceived environment. In a more general form the system can be used to differentiate between actions. In a more specific form it can be used to learn complex mapping functions. We will show three different experiments. The first one shows how to learn actions as a representation of effects. The second one shows how our system can be used to learn a complex mapping function on robot movement and in the third experiment, we illustrate how to combine these independently learned systems to achieve more complex tasks.**

I. INTRODUCTION

Programming by Demonstration (PbD) is an intuitive way to teach new movement skills to a robot. A common view [1] on such approaches classifies them into three increasingly powerful approaches: *mimicry*, *imitation* and *emulation*. The main difference between a robot that is able to *mimic* and a robot that is able to *imitate* is the ability to determine whether an action succeeded or not. Given the according effect is observable by the robot, this can be achieved by determining if an action has caused the expected effect. To equip a robot with the ability to *emulate*, it is additionally necessary to give the robot the ability to achieve an effect by different means than the observed ones. This can be done by using actions from the robots repertoire that cause a similar or equal effect as the observed action.

We propose a system that uses *Bayesian Networks* (BN) to represent actions and their associated effects. After the BN is trained, its probabilistic representation can be used to calculate if an action succeeded. To do that, the action to be executed and its parameters are used to calculate the probable effect the action would have. This result is then compared to the actual observed result after the action was executed. Greater dissimilarities between the real and calculated results can be a hint for a failure. This method serves to achieve *imitation* in PbD. An similar approach can be taken to achieve *emulation*: Instead of calculating the effect, the observed effect is used to infer an action from the robots repertoire that will, according to the BN, create a similar effect.

Allowing a robot to imitate or emulate is only the first step. The next step is to enable the robot to plan. The ability to plan helps to solve complex tasks that depend on the understanding of the task's purpose. When the target is clear, an individual can combine a number of single actions to a sequence in order to solve the task. This is commonly known as planing. To be able to plan, the agent must predict what the outcome of a specific action will be. And so it must for *imitation* (e.g.: When I turn the switch, the light will go on). A planning algorithm will combine a number of actions into a sequence to solve the task at hand. In the sequence, the result of the previous action is the initial state of the next action. The result of the last action in the sequence is equal to the result of the task.

To show the capabilities of our system in simulation, we present three experiments. In the first experiment our system is used to learn the rules of a simple game. Ten cans are used to create a pyramid. Then a ball is thrown at a certain point of the pyramid. The system is trained to predict the result (i.e. number of cans thrown off) of this throw. It is able to infer the outcome for different configurations of the can pyramid and different hit points. In the second experiment an agent throws a ball towards a target point on a wall. The agent has a rather coarse knowledge of how to throw the ball, so that the ball will hit another point. The system will then learn to associate target and actual hit points, which corresponds to a model for the prediction error of this skill. To represent the action, the throw parameters (coordinates of target and hit point) are stored. This experiment will show that using a probabilistic system, like the BN, is a powerful approach for systems that cannot be modeled accurately. The third experiment will combine the independently trained BN from experiment one and two to a task sequence to show how our system can solve more complex tasks. The inference result from system one is used as input for system two.

The organization of the paper is as follows. Related work is discussed in Section II. In Section III the approach in general is introduced. The system is used in Section IV to solve the experiment tasks. Finally in Section V a summary of the results is given.

M. Rudolph and H.-J. Böhme are with the Faculty of Mathematics and Computer Science, University of Applied Sciences Dresden, 01069 Dresden, Germany

M. Mühlig and M. Gienger are with the Honda Research Institute Europe, Carl-Legien-Strasse 30, 63073 Offenbach/Main, Germany.

M. Mühlig is with the Research Institute for Cognition and Robotics, Bielefeld University, 33594 Bielefeld, Germany.

II. RELATED WORK

Programing by Demonstration is a promising approach to equip a robot with actions. It was introduced in [2] by Cypher et al. in the early 1980s. Instead of depending on actions implemented during the development phase, actions are acquired online during a robot's operation. This allows to dynamically enhance the pool of actions by learning new ones, for instance when the system encounters novel situations. A large difficulty however is the implementation of this approach. An interesting survey about the state of the art on different fields in PbD was done by Argall et al. in [3].

One of the different topics in PbD and the focus of this paper is, how to represent the knowledge. In [4] Montesano et al. state that affordance can be described as a relation between actions, effects and objects. They use a Bayesian network to represent this relation. Objects are described by different attributes: size (i.e. small, medium, large), shape (i.e. sphere or box) and color (i.e. blue, yellow, green). The objects can be manipulated using one of the three actions: grasping, touching or tapping. To train the network, objects and actions are combined randomly and the effects are observed. The observed effects are the contact duration between the end-effector and the object, the end-effector velocity and the object velocity. Each attribute and effect and the action are represented with an own node within the network. The relation of the nodes is represented with the edges in the BN.

Bohg et al. present a vision system in [5]. It includes a reasoning system that supports the grasping process in finding the right grasp. This reasoning system is based on the aforementioned approach by Montesano at al. [4]. Bohg extend the system using additional nodes for the context (i.e. environment, task, embodiment) and the cause (i.e. action failure cause). The context is connected to the nodes for objects, actions and effects and the cause is connected only to the effects.

Chella et al. in [6] and Dindo and Infantino in [7] present a Cognitive Framework that uses three different subsystems to learn movements and to generate action plans. The *Subconceptual Area* is used to acquire and process the low-level visual data. The *Conceptual Area* is used to encode and descripe the scene in different spaces. In the *Linguistic Area* a high-level symbolic language is used for representation and processing. The symbols in the linguistic are mapped to appropriate representations of the spaces in the conceptual area.

Toussaint et al. use a probabilistic approach to control a robotic arm in [8]. After they already presented approaches for probabilistic low- and high-level motor control, they show that their approaches not only work in simulated environments but also in real-world scenarios.

In [9] and [10] Mühlig et al. show an approach to represent the movement of an action using *Gaussian Mixture Models*.

Most of the systems that somehow concentrate on features, effects and actions are used to learn object affordances. The term *affordance* was first introduced by Gibson in [11]. Gibson used the term to describe the interaction ability the environment is providing to an individual, depending on the individual's action capabilities and perception. A computer mouse can for instance provide a human with the ability to control a computer; but it can also be used to be thrown somewhere. A primate would probably only perceive the throwing capability.

Kozima et al. [12] propose a system inspired by the mirror system that allows a robot to emulate actions. They state that an action is the manipulation executed with a body movement to produce an effect when applied to an object. It is also proposed that "For the same or similar object o and for the same or similar effect e, there will be a number of different actions a_1,a_2,...". Therefore, when a robot has explored a set of actions it can emulate an observed action by using one of its own actions that produce the same or similar effect on the given object.

III. SYSTEM DESCRIPTION

Our system is inspired by the approach of Montesano et al. [4] who uses a Bayesian network to model the relation between objects, effects and actions. Montesano et al. uses a fixed set of effects, each effect is represented as a discrete node in their network. In our approach we choose to represent the effect implicitly. An effect is the change of one or more features caused by an action. Also, sometimes an effect is not only dependent on one feature but also on other features. It is easier to represent such a multidimensional effect implicitly.

We assume that an action covers a short finite period of time. This means there is a point in time where a set of features represent a scene before an action was executed (i.e. initial state) and after that a set of features that represent a scene when the manipulating ended (i.e. end state). The difference of the features between both states encode the actual feature change. This is only true when the granularity of the action is high enough to exclude any sub goals during the execution of the action.

To account for the two states we use two nodes per feature to represent their changes. One node stores the values of the initial state and the other stores the values of the end state. Because usually a set of features is used, this creates a row of nodes that represent the initial state and another row that represent the end state. The effects itself are represented by the connection between the nodes. To allow the system to potentially learn all possible effects, each node that represents an initial state of a feature is connected to all nodes that represent an end state. The nodes use normal or multivariate *Gaussian Distributions* to store the feature values.

We differentiate between three types of features. (i) Object-features directly describe an object of the observed scene and can hold properties like color, size, weight or visibility. (ii) Meta-features are features that depend on a set of object-features from different objects. An example for meta-features is the distance between objects. The distance is calculated with the absolute difference between the position of two objects. (iii) World-features describe features that do not describe

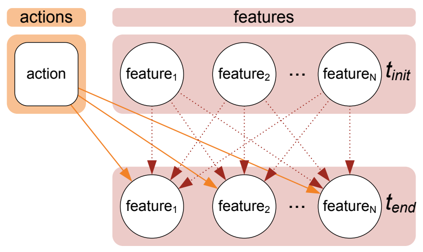

Fig. 1. The general structure to represent actions, objects and effects.

properties of objects but properties of the perceived world in general. For example the overall lightness of an scene or the metered humidity. So instead of the term objects (used by Montesano et al.), we are going to use the term features to describe both, object-related and object-unrelated features.

Finally the action is represented with one discrete node and connected to every end-state node to represent the influence of the action on the features. The resulting BN of these considerations is shown in Fig. 1 where continuous nodes are depicted by circles and discrete nodes by squares.

In the current state the system is able to represent actions using features and effects. This is fine as long as the represented actions do not vary too much in their respective features. For example, in a system that is used to represent the visibility of an object. A node “visible” uses two states, true and false, to represent the object visibility. On applying the action “remove“, the feature ”visible“ will change from true to false. Because this feature has only two values, there will be two very good distinguishable value clusters. Another system is used to represent the position of a ball. The executable action is ”tap the ball“. The force used to tap the ball is randomly distributed over an interval. That will result in an equally random distribution in the observed feature ”position“.

When those sample systems are then used to infer data, the ”remove“-system will do just fine. But when using the ”tap“-system to infer the end position of a ball after it was tapped with a specific force, the system will return the average over all end positions regardless of the applied force, thus resulting in very poor predictions of the ball position. One way to solve this problem is to introduce parameter bound actions (e.g. tap with $F = 10N$, tap with $F = 20N$,...). This solves the problem to the granularity of the discretized parameter, but not for continuous values. To solve this problem, we choose the approach to implement an extra continuous node for the parameter. The node allows to continuously adjust the applied force, which in turn allows a more accurate inference of the position when the system is trained properly.

To represent the parameter in a network structure, an approach with two nodes is chosen. The first node is representing the actual parameters that can be manipulated. The second node represents a feature that is directly influenced by the new parameter. Taking the example of the ”tap“-action, the parameter is representing the force vector that was used to tap the ball and the feature node represents the distance the ball has traveled as a result of the tap.

A more general problem when it comes to *Gaussian distributions* is that it is not capable of representing data that is wide and equally spread over a relative large value interval. To solve this problem we introduce an additional discrete node that is equidistantly clustering the parameter value. This creates defacto a Mixture of Gaussians that can represent a mapping function. The network is basically learning the mapping function for each cluster category. This is shown in the second experiment.

To conduct our experiments, we use a simulation software that combines Open Scene Graph for displaying objects and the PhysX engine to calculate the physical aspects. In the first experiment, called ”can“-experiment, a ball is thrown point-blank at a pyramid of cans. The correlation between impact point and fallen off cans is recorded and used to train the system. In the second experiment, called ”throw“-experiment, a humanoid robot is simulated using the forward chain kinematics approach to control its body. The robot throws a ball. The throwing parameters are captured. The important information are, the target point (i.e. the point the robot aims at) and the hit point (i.e. the ball actually hits). The system is used to learn the mapping function between those two points.

In detail our discretized parameter approach works as follows. In the ”throw“-experiment the target point that is necessary to hit a given hit point is to be inferred. The hit point is used as evidence to infer the target point using the probabilistic abilities of the Bayesian network. First the discrete target point (i.e. category) is inferred. The discrete target point is the cluster ID-number of the equidistantly clustered continuous target point. The result is a probability for each cluster category that states how likely it is that a target point within the range of the selected cluster will result in a hit at the asked point. The most probable grid field is taken and added to the evidence. Only then, the continuous target point is inferred.

It would be possible to implement the parameter approach in the existing structure as shown in Fig. 1. The disadvantage of specializing the network towards the parameter is that it now lacks in scaling to a greater number of actions. Especially when dealing with actions that do not use that specific parameter or parameter feature. The more general network, has the ability to differentiate between a great number of actions. Because as long as a feature is observable it should not matter if it yields any influence on the action or not, the network will learn the influence and compensate for non-influential features. The important point is that all feature data is available for all actions. Data of a parameter is only available when the action uses that parameter.

The BNs are implemented using *Matlab* and the *Bayesian Net Toolbox* (BNT). The Toolbox is presented by Murphy in [13]. To train the BNs the BNT provides a parameter learning function that uses *Maximum Likelihood Parameter Estimation* (see [14]) to adjust the weight parameters along the connection. To infer data, the toolbox provides a implementation of the *Junction Tree* (JT) algorithm [15]. The JT algorithm

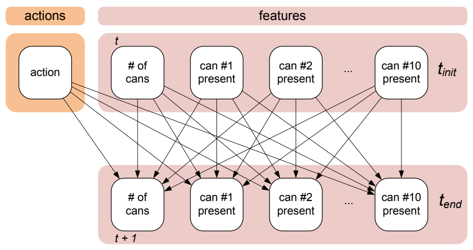

Fig. 2. The *Bayesian network* used to store the can throwing results.

reduces the complexity of the BN to allow to compute the joint probability with less effort.

IV. EXPERIMENTS

A. Can Tossing

The "can"-experiments is created to show our system's ability to learn the rules of an unknown system by observing its behavior upon given actions. Ten cans are stacked up in a triangular shape each can is $16cm$ height and $12cm$ in diameter. The free space between two cans is $2cm$. In the first step of this experiment all physically possible permutations of can combinations were explored. 10 cans that possibly can be in the pyramid or not, allow for $2^{10} = 1024$ potential permutations. But not every of this permutation is physically possible. Each permutation is simulated and accepted as physically possible if the stacked cans did not move more then $1cm$ in any direction within 5 seconds of physical simulation. This resulted in 108 possible combinations. Then a grid of 195 fields is laid over the cans. In this context we define an action as the hit of a ball in the center of one of these fields. Therefore this experiment has 195 actions.

To create samples, each action was executed three times for three different angles over all of the 108 possible can permutations. The angles are $30°$, $0°$ and $-30°$ around the y-axis. The different angles are used to determine the influence of the angles on the resulting can permutation. For each sample the number of cans before and after the throw is recorded. It is also recorded which can left its start position (was thrown off) and which did not. A can is thrown off when it has moved more than $1cm$ in z-direction (i.e. up/down), more than $10cm$ in x-direction or more than $40cm$ in y-direction (i.e. left/right). The ball is thrown with a velocity of $7\frac{m}{s}$ in x-direction towards the cans. In the case of throwing the ball with an angle of $30°$ it has an additional velocity of $-4\frac{m}{s}$ in y-direction. In the case of $-30°$ this velocity is $4\frac{m}{s}$. In all cases the ball has a weight of $2kg$ and hits at the center of the field.

The network to learn the can-throwing rules (Fig. 2) solely consists of discrete nodes because all features are discrete and inferring from discrete nodes is faster. In the first experiment, the system was used to calculate the action that is necessary to throw a given number of cans starting from any given can setup in order to evaluate its planing ability. From the 108 possible can combinations the first, the one with no cans, was left aside. For all other permutations, one action that leaves the setup unchanged and all actions that somehow change the

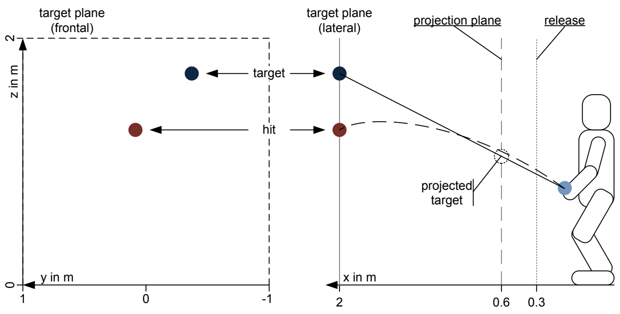

Fig. 3. A sketch of the "Throwing" experiment.

number of cans were calculated. The evidence used to infer the action are the initial number of cans, their position and the desired number of cans.

Due to the fact that the whole can-throwing system is almost deterministic, that the BN uses only discrete nodes and since calculating the inference in this system is merely reading the numbers from a (conditional probability) table, it is expected that the results of the evaluation are very good.

For this evaluation there were 8603 possible combinations. Those combinations were tested in the simulation using the exact parameters for throwing the ball at an angle of $0°$. The BN was only trained with the sample data where the ball also hit at an angle of $0°$. In about 90 percent of the cases, the simulation produced the same outcome as the system predicted. In less then 7 percent the prediction was off by one can either to many or to little. In about 3 percent it was off by more than one can. The detailed results are depicted in Table I where the accuracy is defined with the difference of the number of cans that were to target and the number of cans that were actually thrown off. A negative accuracy means that more cans were thrown off than targeted.

TABLE I
RESULTS FOR THE FIRST EVALUATION ON THE CAN-TOSSING SYSTEM.

accuracy	# of cases	% of all samples
-3 or less	36	0.41
-2	63	0.73
-1	309	3.59
0	7780	90.43
1	269	3.12
2	113	1.31
3 or more	33	0.38

B. Throwing

A "throwing"-experiment was created to test the ability of our system to learn the complex mapping function between target point and hit point. We show that it is possible to learn a non-linear mapping function between two two-dimensional continuous values. As mentioned briefly in Section III, we use a virtual humanoid robot to throw a ball. The robot is placed at the origin of the simulated environment. The robot is then used to throw the ball at a target point that is located at the target plane. The target plane is located $2m$ in front (i.e. x-direction) of the robot. The situation is sketched in Fig. 3. The throwing motion starts at a fixed point and moves the hand holding the ball straight towards the target point without violating the joint

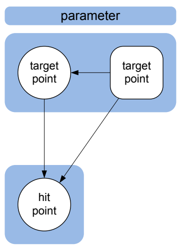

Fig. 4. The *Bayesian Network* used to store the throwing result. (circles = continuous, squares = discrete)

limits of the limb. The ball is released $30cm$ in front of the robot and continuous its flight towards the target plane with the velocity it inherited at the release point. Eventually the ball will cross the target plane. The point where the ball crosses the plane is defined as the hit point. The target point lies within an area on the target plane. This area is $2m$ in height and width and its center is aligned with the robot's center. 30.000 samples were created throwing the ball using randomly chosen target points and recording the hit point.

The system (Fig. 4) was trained with 29.000 samples, leaving 1.000 samples for evaluation purpose. Because of the of the robot's joint limits, it is not able to actually throw towards all target points. The restrictions also result in a non-linear mapping function which is not feasible with only one *Gaussian distribution*. As described before, we us an additional discrete node to cluster the target point and thereby creating a *Mixture of Gaussians* to enable the system to cope with the mapping function. To evaluate the system, the not-learned sample data is used as evidence to infer the target point. To infer the target point, first the discrete target point category is inferred and added to the evidence. In the next phase, the continuous target point is inferred. Then the data it is used in the simulation to thrown the ball accordingly. The result of the throws is then compared to the data from the evaluation samples and the error, calculated using the RMS-error. The resulting error in x-direction is about $0.0178m$ and in y-direction about $0.0305m$.

C. Combining Throwing and Cans

Finally we want to show that both systems can be combined to work together to simulate a simple planning system. In this combined system, the result of the "can"-system is used as input for the "throw"-system. Both systems were trained independently and to show that they can be combined without specific adaptation. By combining a number of independent subsystems, a larger planing system could be created. It is only necessary to create an additional controlling system that decides which task to solve. In this example the systems are used to create a given can configuration.

For this experiment the "can"-network was trained using the data with an impact angle of $-30°$ because these data seems more similarly to the impact angle of an throw from the humanoid than the other angles. Still there is no variation for the angle around the z-axis. Therefore it is expected that accuracy on the right side of the can pyramid will be better then on the left side. This is because the robot uses its right arm to throw the balls and the further the robot aims to the left the bigger the lateral impact angle will get.

To connect the systems, the inferred action of the "can" system, which represents a hit point, is taken as point to hit to the "throw"-system. There it is used to infer the target point. To evaluate this system, different scenarios were created.

In the first scenario, the system infers actions to throw off the top three cans of the ten can pyramid. To do so, the system first tries to find actions that would lead to the desired result with one throw. If no action is available it creates intermediate goals by adding cans to the final goal till an action can be inferred. As described the inferred actions from the "can"-system are used to infer the target point with the "throw"-system. The resulting data is then evaluated in the simulation. The possible result can be (i) a success, (ii) a fail because to many cans have been thrown off, (iii) a fail because no can was hit, (iv) the creation of an intermediate state because not enough cans were thrown off. In the case of (iv) the resulting can configuration (i.e. intermediate state) is used as new initial state to infer new actions to reach the target configuration. If the same intermediate state is generated multiple times during the whole test, it still is only evaluated once in the simulation.

In this first scenario six actions were created in the first inference round and were then simulated. In one case each the throw failed because of (ii) and (iii). In two cases the system succeeded immediately and in two cases it did not throw off all cans it should (iv). The two (iv) cases created two different intermediate states for the second round. In both cases only one can was left and for both cases seven new actions were created. From these 14 actions 10 succeeded, two failed with (ii) and two failed with (iii). The success rate is calculated by taking the percental share of the cases (i) and adding to it the share of the cases (iv) multiplied with their success rate, thus the overall success rate is about 57.13%. The result is summed up in Table II;

TABLE II
PERFORMANCE OF THE SYSTEM IN THE FIRST SCENARIO.

round	1	2a	2b
cans to throw	3	1	1
actions	6	7	7
success (i)	2	5	6
fail (ii)	1	2	0
fail (iii)	1	0	2
interm. (iv)	1 (to 2a) 1 (to 2b)		
success rate	57.13%	71.42%	71.42%

The second scenario is used to compare the accuracy when throwing at the right and left side of the pyramid. Therefore the four cans on the right or left flank were to be removed respectively. First the throwing at the four cans at the right side, the side directly in front of the robot's throwing arm, is evaluated. The results are shown in Table III. To achieve the target, the system calculated eleven actions. Four throws created two different intermediate configurations. For the first intermediate state, twelve new actions were create to remove one can. The second intermediate state left three cans to be removed. 29 actions to remove these cans were found. 14 actions lead back to intermediate state 2a and seven actions created a new intermediate state with two cans left to remove. The last state spawned 17 actions. The overall success rate is about 68.57%.

TABLE III
PERFORMANCE OF THE SYSTEM FOR THE FIRST PART OF THE SECOND SCENARIO.

round	1	2a	2b	3
cans to throw	4	1	3	2
actions	11	12	29	17
success (i)	5	8	0	14
fail (ii)	0	0	1	1
fail (iii)	0	4	7	2
interm. (iv)	3 (to 2a) 1 (to 2b)		14 (to 2a) 7 (to 3)	
success rate	68.57%	66.66%	51.65%	82.00%

When throwing at the left side of the pyramid, the success rate declines. The complete result is displayed in Table IV. In the first round of inferring actions, eleven actions were created to throw off the four cans. Four actions created three new intermediate states with a various number of cans left to throw. In the third round one last intermediate state is created. The overall accuracy is about 57.71%.

TABLE IV
PERFORMANCE OF THE SYSTEM FOR THE SECOND PART OF THE SECOND SCENARIO.

round	1	2a	2b	2c	3
cans to throw	4	3	1	2	2
actions	11	29	36	53	17
success (i)	4	0	14	18	14
fail (ii)	2	2	8	7	1
fail (iii)	1	7	14	10	2
interm. (iv)	1 (to 2a) 1 (to 2b) 2 (to 2c)	14 (to 2b) 6 (to 3)		18 (to 2a)	
success rate	57.71%	35.79%	38.88%	47.16%	82.35%

In conclusion of the second evaluation scenario, it is clearly visible that the lateral angle is influencing the result. As for the combination of both the "throw"- and the "can"-system, even though they were trained independently from each other, both system can be combined to solve the task. The results are good but not perfect and can probably be improved by integrating more systemic knowledge.

V. CONCLUSIONS

We have shown an approach to use a BN to store actions as a representation of their feature changes by using a specific structure. We have then shown a system that is able to learn a complex mapping function by incorporation of parameters of an action. The first part of the system was tested by learning the correlation between the point where to hit a can pyramid and the number of cans that fall off. The problem was rather deterministic due to the use of discrete values and the repeatability of the physical simulation. After testing the system, it showed a low prediction error. The error can be explained by minor deviations in physical simulation. The parameter network was used to learn the mapping between the point a simulated humanoid robot has to aim when throwing a ball and the actual hit point.

Then both systems were combined to throw at the cans using the simulated robot. Despite the fact that both systems were designed and trained independently and without adjusting one system to the other, it showed good results. The combination of both system shows that a number of such systems, independent on their specific task, can be used to create a planning system. Such a system can be used to plan over a number of actions in order to achieve a certain task.

In the future we want to integrate our parameter network in our action and feature representing network. To increase the accuracy of our combined system, different robot positions and throwing actions should be evaluated. We also we want to transfer our system to a real robot and adapt it to movement learning.

VI. ACKNOWLEDGMENTS

M. Rudolph thanks the Honda Research Institute Europe for the means to accomplish this work.

REFERENCES

[1] C. Breazeal and B. Scassellati, *in Imitation in Animals and Artifacts*. MIT Press, 2002, ch. Challenges in Building Robots That Imitate People.

[2] A. Cypher, D. C. Halbert, D. Kurlander, H. Lieberman, D. Maulsby, M. B. A., and A. Turransky, Eds., *Watch what I do: programming by demonstration*. Cambridge, MA, USA: MIT Press, 1993.

[3] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robot. Auton. Syst.*, vol. 57, no. 5, pp. 469–483, 2009.

[4] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor, "Learning object affordances: From sensory motor coordination to imitation," in *Transactions on Robotics*, 2007.

[5] J. Bohg, C. Barck-Holst, K. Huebner, B. Rasolzadeh, M. Ralph, D. Song, and D. Kragic, "Towards grasp-oriented visual perception for humanoid robots," *International Journal on Humanoid Robotics*, vol. 6, no. 3, pp. 387–434, September 2009.

[6] A. Chella, H. Dindo, and I. Infantino, "A cognitive framework for imitation learning," *Robotics and Autonomous Systems*, vol. 54, pp. 403–408, 2006.

[7] H. Dindo and I. Infantino, "Representation, recognition and generation of actions in the context of imitation learning." in *EUROS*, ser. Springer Tracts in Advanced Robotics, H. I. Christensen, Ed., vol. 22. Springer, 2006, pp. 65–77.

[8] M. Toussaint, N. Plath, T. Lang, and N. Jetchev, "Integrated motor control, planning, grasping and high-level reasoning in a blocks world using probabilistic inference." in *IEEE International Conference on Robotics and Automation*, 2010.

[9] M. Mühlig, M. Gienger, S. Hellbach, J. J. Steil, and C. Goerick, "Task-level imitation learning using variance-based movement optimization," in *Proc. IEEE International Conference on Robotics and Automation (ICRA 2009)*, 2009.

[10] M. Mühlig, M. Gienger, J. J. Steil, and C. Goerick, "Automatic selection of task spaces for imitation learning," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems.*, 2009.

[11] J. J. Gibson, *The Theory of Affordances*, R. Shaw and J. Bransford, Eds. Lawrence Erlbaum, 1977.

[12] H. Kozima, C. Nakagawa, and H. Yano, "Emergence of imitation mediated by objects," pp. 59–61, 2002.

[13] K. P. Murphy, "The bayes net toolbox for matlab," October 2001.

[14] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2007, ch. 2, p. 93ff.

[15] C. Huang and A. Darwiche, "Inference in belief networks: A procedural guide," *International Journal of Approximate Reasoning*, vol. 15, no. 3, pp. 225–263, 1996.