Learning Models for Constraint-based Motion Parameterization from Interactive Physics-based Simulation

Zhou Fang[1,2], Georg Bartels[1] and Michael Beetz[1]
{yfang, bartelsg, beetz}@cs.uni-bremen.de

***Abstract*— For robotic agents to perform manipulation tasks in human environments at a human level or higher, they need to be able to relate the physical effects of their actions to how they are executing them; small variations in execution can have very different consequences. This paper proposes a framework for acquiring and applying action knowledge from naive user demonstrations in an interactive simulation environment under varying conditions. The framework combines a flexible constraint-based motion control approach with games-with-a-purpose-based learning using Random Forest Regression. The acquired action models are able to produce context-sensitive constraint-based motion descriptions to perform the learned action. A pouring experiment is conducted to test the feasibility of the suggested approach and shows the learned system can perform comparable to its human demonstrators.**

I. Introduction

An autonomous service robot has to perform manipulations with requirements that go beyond contact-free point-to-point motions. Tiny variations in *how* a motion is executed in a certain context can mean the difference between success and failure. For instance, consider making breakfast: how easy is it to accidentally destroy a sunny side up egg when removing it from the pan, or to spill milk while pouring from a full pack? To perform such actions robustly, the control program requires models that relate motions executed in a particular task context to the resulting effects, and vice versa. For example, they describe how transporting a pancake from pan to plate depends on the geometric and dynamic properties of objects, their positions in the environment, etc. Such models enable robots to adapt their motion execution to achieve the desired effects.

Equipping robots with this kind of models is very attractive because it would enable developers to program their robots at a higher level of abstraction. Instead of specifying motion trajectories, they can instruct robots at the level of sequences of object property changes. The control program would context-sensitively perform the right motions with the available tools, self-selecting the appropriate motion parameters. In other words, acquiring action models that inform robots how to perform a motion to achieve certain effects is a key milestone on our path towards autonomous service robots.

Unfortunately, the research on design and acquisition of action knowledge relating effects and robot motion is still in its early stages. Despite great progress in the fields of task planning [1], [2], knowledge representation [3], [4] and learning by demonstration [5] for robots, our community lacks quantitative and qualitative representations of object property changes such as spilled milk or misshaped pancakes that can be reasoned about in terms of *how* they were achieved or *how* they can be undone. Other components, such as reliable, context-sensitive perception of effects in a variety of (household) environments, are also needed. However, even if we did have optimal knowledge representation for and perception of motion effects, we would not know how to use these modules to context-sensitively execute the appropriate robot movements.

In prior work we proposed to use physics simulation environments as a tool for acquiring knowledge about object manipulation tasks. There are two major benefits to this approach: (1) complete perception: the entire world state is accessible and (2) scalability: human demonstrations can be crowd-sourced online and large amounts of autonomous projections in different scenario's can be generated.

In this paper, we present a framework for acquiring robotic action knowledge from user demonstrations in an interactive simulation environment with realistic physics. We present an experiment in which we learn an action model that predicts the correct constraint-based movement descriptions for pouring pancake batter given certain environment properties. Because the models specify the nature and existence of the relationship between an environment variable and motion parameters, it can be used to derive other forms of (generalized) knowledge as well.

Our contributions beyond [6]–[8] are as follows. First, we acquire action models that context-sensitively select constraint-based motion descriptions. These descriptions can be used for advanced whole-body motion control of robots [9], and have been incorporated into formal robot knowledge bases [10]. Kunze *et al.* [6] acquired action models from simulation, but the underlying motion representations and controllers were less sophisticated. Their system focused on predicting (symbolic) effects rather than generating action and did not attempt to learn the models from humans. Kunze *et al.* [7] and Haidu *et al.* [8] used demonstrations from users to learn action knowledge, but did not apply this knowledge to performing actions. Furthermore, we demonstrate the feasibility of acquiring these action models from demonstrations of naive users, i.e. users unaware of neither the learning technique employed nor the controllers used for evaluation.

[1]The authors are with the Institute for Artificial Intelligence, Universität Bremen, Am Fallturm 1, 28359 Bremen, Germany.

[2]Corresponding author.

This work is supported by the EU FP7 project *RoboHow* (Grant Agreement Number 288533).

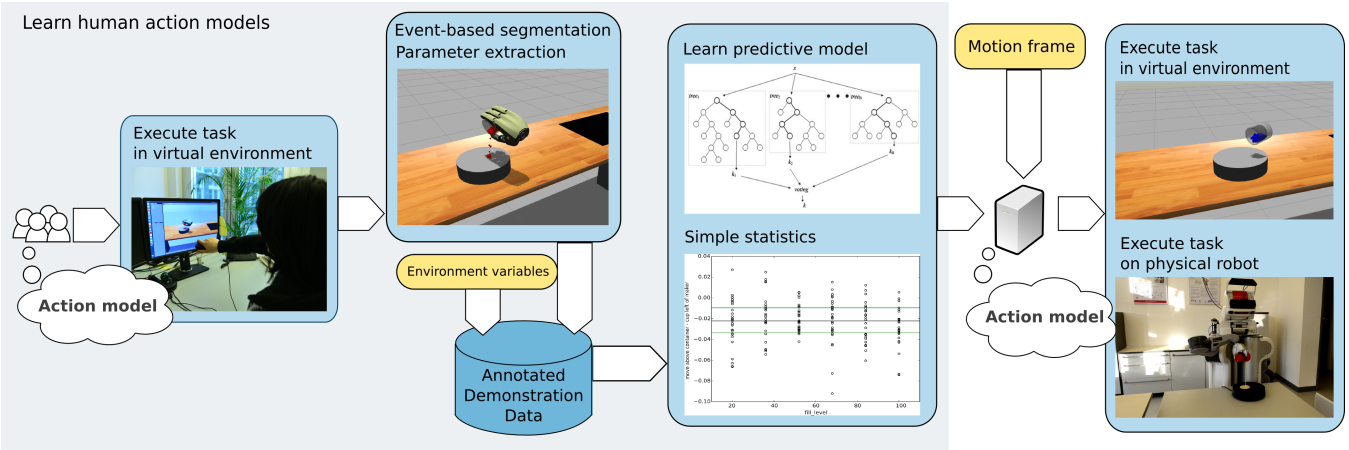

Fig. 1. System overview

II. Overview

An overview of the system is depicted in Figure 1. Naive demonstrators perform a task in a simulator under various circumstances. The complete simulation data is recorded. The system segments the demonstrations using events and event intervals. Environment variables and poses are extracted at the key timepoints as defined by the events and are used to train a set of models that predict motion constraints based on environment variables. The models are automatically evaluated to establish whether the observed variation in the environment affects the motion constraints, which constraints are affected, and how. Presumably human adults are able to perform an action reliably using various objects in various conditions because they possess a model relating motions to action effects. By studying not only the motions, but also how changes in these motions relate to changes in the environment and action effects, our system attempts to derive the relevant implicit knowledge people have about the task and the world. When the system is tasked with performing the learned action, it uses the observed environment variables and learned models to instantiate the controllers appropriately.

The components of system are described in more detail in Section III and the instantiation thereof for the experiment is described in Section IV.

III. Methodology

Before explaining our methodology, we briefly introduce the mathematical notations used throughout this paper.

- Roman lowercase letters (a) denote scalars, bold lowercase letters ($\mathbf{b}$) denote column vectors, and bold uppercase letters ($\mathbf{C}$) denote matrices.
- Subscripts denote indexes and are used to point to elements of corresponding vectors and matrices. E.g., a_i is the i-th element of vector $\mathbf{a}$, while $A_{i,j}$ is the element in i-th row and j-th column of $\mathbf{A}$.
- The lowercase letters f and g with arguments specified in trailing brakets denote functions; roman subscript letters are used to denote different functions, e.g. $f_l(\dots)$
- $\mathbf{e}_m$ denotes a column vector containing only 1 as elements with dimensionality m, e.g. $\mathbf{e}_2 = [1, 1]^T$
- ${}^W_A\mathbf{T}$ denotes the pose of coordinate frame A expressed with respect to the coordinate frame W, represented as a homogeneous transformation matrix; ${}^W_A\mathbf{R}$ denotes the corresponding rotation, and ${}^W_A\mathbf{p}$ denotes the corresponding translation of A with respect to W

A. Simulation environment and recording

The interactive simulations were developed using Gazebo Simulator [11], a robotic simulator with realistic physics engine. The user interacts with the simulated environment through the motion sensing game controller Razer Hydra, which calculates its position using a weak magnetic field to a precision of up to 1mm and 1deg. The recorded movements are projected onto a simulated robotic hand. The joystick on the controller is used to close and open the hand for grasping. A game controller was preferred because of the ease of use and availability to the general public. For more details on the simulation, see [12].

The physics data produced by the simulator are recorded at 1000Hz and stored using MongoDB. This data is a complete representation of what happened during the demonstration.

The recorded demonstrations are then segmented and processed to provide an appropriate dataset. There are many approaches for segmenting observed actions, such as looking at the (in)variability of certain parameters over trials and time, cluster trajectory points, and recognition based on previously defined prototypes [13]–[17]. Here we chose to use event detectors to segment the action into subactions. Aksoy *et al.* [18] demonstrate that using only a couple key events (changes in object relationships), actions can be identified reliably. Such an approach allows actions (and thus demonstrations thereof) to be represented compactly. The used event detectors indicate the intervals during with their respective events have occurred. A small set of detectors for basic events such as contact and grasping can already describe a large set of actions. Action recognition and segmentation is achieved by matching event intervals to the expected pattern.

B. Motion Control Framework

The framework for constraint-based motion control we employ is a simplification of [9]. For the sake of readability and completeness, we describe our entire framework rather than listing the differences. The framework allows one to specify motions as a composition of various types of constraints. Constraints can easily be added to or removed from the motion specification and allow for an efficient specification of complex tasks.

Motion frame

```
define manipulation problem:          pour(pancakemix, pancakemaker)
  objects
    Pancakemaker:                     cylinder_mesh1
      Frames of reference:            top-center[0,0,0]
    Cup:                              cylinder_mesh2
      Frames of reference:            top-center[0,0,0],
                                      bottom-center=[top-center+[0,0,0.108]]
  motionphases:
    1:                                move-above-pancakemaker
      max-trans-velocity:             <...>
      min-trans-velocity:             <...>
      task-constraints:
        cup-bottom-behind-maker:      <..., ...> (min, max)
        mug-bottom-left-maker:        <..., ...> (min, max)
        mug-bottom-above-maker:       <..., ...> (min, max)
        mug-top-behind-bottom:        <..., ...> (min, max)
        mug-top-left-bottom:          <..., ...> (min, max)
        mug-top-above-bottom:         <..., ...> (min, max)
        ...                           ...
    2:                                tilt-bottle-down
      ...
    3:                                tilt-bottle-back
      ...
```

Fig. 2. Motion frame for a pouring task

A sample motion specification from the domain of pancake making is given in Figure 2. It shows a task description composed of three motionphases, each corresponding to a controller which tries to satisfy a particular set of kinematic constraints. Due to space restrictions not all constraints are listed. Here the object definitions pancakemaker and cup are used, but the framework is applicable to different domains.

Each of the motion controllers can be viewed as a mathematical function

$$\dot{\mathbf{c}}_{des} = f(\mathbf{o}), \tag{1}$$

with $\mathbf{o}$ denoting a vector of *observable variables* $\mathbf{o} \in \mathbb{R}^n$, and $\dot{\mathbf{c}}_{des}$ representing the desired instantaneous change of a vector of *controllable variables* $\mathbf{c} \in \mathbb{R}^m$. We assume that $\mathbf{c} \subseteq \mathbf{o}$ holds. $f(\mathbf{o})$ is called in every control cycle.

At the heart of $f(\mathbf{o})$ lies a minimization problem which is constructed from two types of constraints: *Controllable constraints* bounding the *optimization vector* $\mathbf{s}$, and *task constraints* limiting the desired instantaneous change of a vector of *task functions* $\mathbf{e} = f_e(\mathbf{o})$, with $\mathbf{e} \in \mathbb{R}^p$.

Using a vector of *slack variables* $\boldsymbol{\epsilon} \in \mathbb{R}^p$, related to the p task constraints, we designed $\mathbf{s}$ as

$$\mathbf{s} = [\dot{\mathbf{c}}_{des}^T, \boldsymbol{\epsilon}^T]^T. \tag{2}$$

Each slack variable in $\boldsymbol{\epsilon}$ is associated with one task constraint. The solver of the underlying minimization problem uses $\dot{\mathbf{c}}_{des}$ to move the actual robot, while it employs $\boldsymbol{\epsilon}$ to violate task constraints which are uncovertible or too costly to obey. The solver cannot violate controllable constraints.

The minimization problem solved in $f(\mathbf{o})$ is then:

$$\min_{\mathbf{s}} \quad \mathbf{s}^T\mathbf{H}\mathbf{s} \tag{3a}$$

$$s.t. \quad \mathbf{l}_A < \mathbf{A}\mathbf{s} < \mathbf{u}_A \tag{3b}$$

$$\mathbf{l} < \mathbf{s} < \mathbf{u} \tag{3c}$$

Equation (3a) defines the *cost function* of the problem, while equations (3b) and (3c) define the task and controllable constraints, respectively. $\mathbf{H}$ denotes a square diagonal *weighting matrix* modulating the cost function chosen as

$$\mathbf{H} = diag(\mathbf{w}), \quad \mathbf{w} = [f_{cw}(\mathbf{o})^T, f_{tw}(\mathbf{o})^T]^T. \tag{4}$$

$f_{cw}(\mathbf{o}) \in \mathbb{R}^m$ and $f_{tw}(\mathbf{o}) \in \mathbb{R}^p$ compute the *controllable weights* and *task weights* in every control cycle, respectively. We assume that $\forall i : w_i \geq 0$. Informally speaking, the higher a controllable weight the more costly it is to employ the corresponding controllable variable, and the higher a task weight the more costly it is to use its slack variable.

The boundaries on the desired instantaneous change of the task functions are calculated in every control cycle as:

$$\mathbf{l}_A = f_{lA}(\mathbf{o}), \quad \mathbf{u}_A = f_{uA}(\mathbf{o}). \tag{5}$$

The relationship between the optimization vector $\mathbf{s}$ and the task constraints is analytically calculated in every iteration using the task functions $f_e(\mathbf{o})$:

$$\mathbf{A} = \begin{pmatrix} \mathbf{J} & \mathbf{I}_{p\times p} \end{pmatrix}, \quad \mathbf{J} \in \mathbb{R}^{p\times m}, \quad J_{i,j} = \frac{\partial e_i(\mathbf{o})}{\partial c_j} \tag{6}$$

Finally, the boundaries on $\mathbf{s}$ are set in every control cycle as the computed values $f_l(\mathbf{o})$ and $f_u(\mathbf{o})$ for $\dot{\mathbf{c}}_{des}$, and as fixed values for $\boldsymbol{\epsilon}$:

$$\mathbf{l} = [f_l(\mathbf{o})^T, -\infty\mathbf{e}_p^T]^T \quad \mathbf{u} = [f_u(\mathbf{o})^T, +\infty\mathbf{e}_p^T]^T \tag{7}$$

It follows that all slack variables $\boldsymbol{\epsilon}$ are effectively unbounded. Hence, the slack variables allow the solver of the minimization problem to completely ignore the desired change of any task constraint if it is uncovertible or too costly. On the other hand, in case an underconstrained minimization problem is given, i.e. there are redundant controllable variables, the solver calculates a solution with minimal costs.

In summary, to specify a controller in our framework one has to define $\mathbf{o}$ and $\mathbf{c}$, and select compatible functions $f_{cw}(\mathbf{o})$, $f_{tw}(\mathbf{o})$, $f_{lA}(\mathbf{o})$, $f_{uA}(\mathbf{o})$, $f_l(\mathbf{o})$, $f_u(\mathbf{o})$, and $f_e(\mathbf{o})$.

C. Learning Framework

To instantiate the controller, we designed a simple learning module that uses Random Forest Regression models [19] to predict the task constraints the controller must satisfy to execute a given action successfully given the observed environment variables.

Parameters are extracted from the recorded interactive simulations at key timepoints using event detectors to form a training dataset $\mathbf{D}$ of N demonstrations. $\mathbf{D}$ consists of $\{x_i\}_{i=1}^N$ environment variables and $\{g_i\}_{i=1}^N$ motion goal variables. It is assumed that the task-specific constraints for the motion controller to replicate the task can be derived from these motion goal variables.

The learned models should approximate a function $\mathbf{g} = f(\mathbf{x})$ that describes the relation between the *environment variables* $\mathbf{x}$ and the *motion goal variables* for the task $\mathbf{g}$, as an approximation of the knowledge that the demonstrators possess of this relationship.

Regression forests use an ensemble of O regression trees, where each regression tree is constructed using a bootstrap sample from $\mathbf{D}$, a randomly drawn set with replacement equal to N, and a random subset of all possible features k. At each node the best split for the tree is determined from this subset of features.

Note that the exact choice of the number of trees O is generally considered not crucial as long as it is large enough. I.e. in principle the larger the number of trees the better, but the improvement in results becomes limited beyond a certain number and may not outweigh the additional computational cost [20], [21]. The appropriate amount of trees is difficult to estimate and depends on the characteristics of the training set such as the number of training samples available.

By averaging over these trees with random training subset and random feature subset, the forest as a whole is able to reliably model the relationship between $\mathbf{g}$ and $\mathbf{x}$. The trained model can then be used to predict $\mathbf{g}$ for novel $\mathbf{x}$.

It is to be expected that not all g_i will depend heavily on $\mathbf{x}$, as certain aspects of a task can be relatively invariant over different contexts/outside factors. Therefore, the learning module determines for each constraint variable whether it can be reliably predicted based on the training results. It does so using an estimation of the variance explained by the model for that variable. If a given threshold is not exceeded, a static range based on $\mathbf{D}$ is used instead of the value predicted by the model.

IV. Evaluation

A. Experiment setup

A pouring task is considered to illustrate the validity of the proposed approach. Six naive participants were asked to pour pancake mix onto a pancakemaker in the simulation environment. The fullness of the containers was varied in 6 conditions: 20, 36, 52, 68, 84 or 100 percent of the container volume was filled with liquid. Liquid was simulated using small spheres. The position of the cup was varied randomly (while still being reachable and on the table). The participants performed one practice block and six experimental blocks, with each block consisting of 5 pouring repetitions. Subjects were instructed to do an additional demonstration for each demonstration of which they thought it might have failed, to be certain that there would be enough good demonstrations in the set. Demonstrations may fail for a number of reasons; the subject might spill some liquid or there may be a technical problem with the simulator. A Latin square design was used to counterbalance condition order [22]. while obtaining a fair sample of demonstrations.

The simulation is initialized with a pancakemaker and a cup on the table. The user acts in that environment using the motion sensing game controller. Every participant was shown the same instruction sheet informing them of the task, the number of blocks, and demonstrations per block. They were not informed about the purpose of the experiment or the changing conditions. The participant had to grasp the cup using the virtual hand, move it above the pancakemaker, tilt it so that enough liquid for a small pancake (ca. 75 spheres, equivalent to 54.75ml) would flow out and then put the cup back on the table and release it. They were given one practice block of 5 repetitions or more, until they felt comfortable performing the task in the simulator.

B. Event segmentation and variable extraction

The demonstration data contains the pose and contacts of each object at each timepoint. At each time point, the detectors indicate whether their event has occurred. These event points are combined to form event intervals. The pouring demonstrations were segmented using three detectors, detecting lifts (object is not supported by any surface), rotation of the controlled object, and particles leaving the container. We expected pouring actions to follow the pattern of $lift_{starttime} < rotation_{starttime} < leaving_{starttime} < leaving_{endtime} < rotation_{endtime} < lift_{starttime}$. There could be several rotation intervals in a pouring action, but there should be one rotation starting around the beginning of "leaving" and one ending around the end of "leaving".

We assumed a fixed structure for the pouring task as a prior for learning the action models, consisting of three motion-phases: *move container above pancakemaker*, *tilt container down*, and *tilt container back*. The first phase was assumed to span the interval during which the container was lifted until pancakemix started leaving the container. The second phase spanned the time during which the pancakemix was leaving the container, and the third phase from when the mix stopped leaving the container until the end of the lift. The associated constraint variables were extracted from these time intervals, resulting in a light representation of every demonstration that could be used by the learning module.

C. Modelling controllers for pouring motions

The structure of the controllers for the pouring motions were designed to match the same three phases as described above. Each phase was formalized within the motion control framework (Section III-B).

In our simulation there were three relevant reference frames: The *world* frame, a frame attached to the bottom center of the pouring container (called *cup*), and a frame attached to the top center of the pancakemaker (called *maker*). In subsequent notations we abbreviate all three frames to their uppercase first letter. Please note that both relevant objects are rotational symmetric, and that both *C* and *M* are positioned with their origins on and their z-axes aligned with the symmetry axis of their respective objects.

We parametrized observable homogeneous transformations between reference frames by a set of observable variables $\boldsymbol{\chi} = [x, y, z, \alpha, \beta, \gamma]^T$. We translated a given $\boldsymbol{\chi}$ into a transformation matrix using

$$\mathbf{T}(\boldsymbol{\chi}) = \begin{pmatrix} \mathbf{R}_z(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_x(\gamma) & [x, y, z]^T \\ \mathbf{0}_{1\times 3} & 1 \end{pmatrix} \quad (8)$$

In every control cycle, we observed the homogeneous transformation matrices ${}^{W}_{C}\mathbf{T} = \mathbf{T}(\boldsymbol{\chi}_{cup})$ and ${}^{W}_{M}\mathbf{T} = \mathbf{T}(\boldsymbol{\chi}_{maker})$, and declared the observable and controllable variables of our pouring controllers as

$$\mathbf{o} = \boldsymbol{\chi}_{cup} \cup \boldsymbol{\chi}_{maker} \qquad \mathbf{c} = \boldsymbol{\chi}_{cup} \tag{9}$$

We modelled computed the lower and upper boundaries of the controllable constraints for all three motion phases as

$$f_{u_{1/2/3}}(\mathbf{o}) = [v_{min_{1/2/3}}\mathbf{e}_3^T, \omega_{min_{1/2/3}}\mathbf{e}_3^T]^T \tag{10}$$
$$f_{l_{1/2/3}}(\mathbf{o}) = [v_{max_{1/2/3}}\mathbf{e}_3^T, \omega_{max_{1/2/3}}\mathbf{e}_3^T]^T \tag{11}$$

We defined three point features for control: the center bottom point of the cup ${}^{W}_{C_B}\mathbf{p} = {}^{W}_{C}\mathbf{p}$, the center top point of the maker ${}^{W}_{M_T}\mathbf{p} = {}^{W}_{M}\mathbf{p}$, and the center top point of the cup ${}^{W}_{C_T}\mathbf{p} = {}^{W}_{C}\mathbf{R} * [0, 0, 0.108]^T + {}^{W}_{C}\mathbf{p}$. Note that the pouring container had a height of $10.8cm$ and a radius of $5cm$.

Using three auxiliary task functions

$$e_1(\mathbf{o}) = {}^{W}_{C_B}\mathbf{p} - {}^{W}_{M_T}\mathbf{p} \tag{12}$$
$$e_2(\mathbf{o}) = {}^{W}_{C_T}\mathbf{p} - {}^{W}_{M_T}\mathbf{p} \tag{13}$$
$$e_3(\mathbf{o}) = {}^{W}_{C_T}\mathbf{p} - {}^{W}_{C_B}\mathbf{p}, \tag{14}$$

we defined the actual task functions for each of the three motion phases as

$$f_{e_{1/3}}(\mathbf{o}) = [e_1(\mathbf{o})^T, e_3(\mathbf{o})^T]^T, \tag{15}$$
$$f_{e_2}(\mathbf{o}) = [e_2(\mathbf{o})^T, e_3(\mathbf{o})^T]^T. \tag{16}$$

The functions computing the lower and upper boundaries of the task constraints for all three motion phases were chosen as

$$f_{lA_{1/2/3}}(\mathbf{o}) = \mathbf{g}_{lA_{1/2/3}} - f_{e_{1/2/3}}(\mathbf{o}) \quad \mathbf{g}_{lA_{1/2/3}} \in \mathbb{R}^6 \tag{17}$$
$$f_{uA_{1/2/3}}(\mathbf{o}) = \mathbf{g}_{uA_{1/2/3}} - f_{e_{1/2/3}}(\mathbf{o}) \quad \mathbf{g}_{uA_{1/2/3}} \in \mathbb{R}^6 \tag{18}$$

Finally, we used the same functions to compute to task and controller weights for all three motion phases. Using the constants $\mu = 0.001$ and $w_t = 1$ in all our evaluation experiments we computed the weights as

$$f_{cw}(\mathbf{o}) = \mu\mathbf{e}_6 \quad f_{tw}(\mathbf{o}) = (w_t + \mu)\mathbf{e}_6 \tag{19}$$

We simplified the control problem by (1) disabling the simulated gravitational force acting on the pouring container, and (2) setting low mass values on the simulated fluid. This makes the effect of gravity acting on the controlled container negligible. Hence, we did not use a feedforward term to correct for it.

D. Learning models

The training set $\mathbf{D} = \{x_i\}_{i=1}^{N} \cup \{g_{i,j}\}_{i=1,j=1}^{N,M}$ consists of $N = 211$ demonstrations with $M = 3$ motionphases, where x_i^k is the k-th environment variable of demonstration i and $g_{i,j}^l$ is the l-th motion constraint variable from motionphase j of demonstration i. $g_{i,j}^l$ corresponds to $\mathbf{g}_{Aj}$ given a certain demonstration i. A sample of the data is shown in Table I.

Each motionphase has 14 contraint parameters, which are not all shown in the table due to space restrictions. The parameters consist of the positions of the three control points (${}^{W}_{C_B}\mathbf{p}$, ${}^{W}_{C_T}\mathbf{p}$, ${}^{W}_{M_T}\mathbf{p}$) relative to each other. The relative position of the top of the cup (${}^{W}_{C_T}\mathbf{p}$) to the bottom (${}^{W}_{C_B}\mathbf{p}$) contains information equivalent to the rotation of the container. Additionally, 4 parameters encode the minimum and maximum rotational and translational velocity during the motionphase. Finally, one parameter describes the waiting time from the current phase to the next.

The Random Forest Regression models are implemented using the scikit-learn package [23]. The forest consisted of 100 trees and the maximum amount of features per tree was set to equal K environment variables. The trees were allowed to fully grow, since bagging methods like Random Forest reduce overfitting and work quite well with complex single models. The environment variables that were varied in this experiment were the volume of the cup and the starting position of the cup on the table.

We assume that a motion can be successful within a certain range of motion constraints, rather than a single value. The various demonstrations are expected to reflect this acceptable variance in execution. The learned models produce a single value for each component of $\mathbf{g}_{Aj}$ however. Therefore, for each component of goal $\mathbf{g}_{Aj}$, the Root Mean Squared Error (rmse) was taken as a reflection of the average difference between the estimator and the estimated value, and therefore giving an indication of the size of the envelope around the estimator based on our samples. The rmse of each component was subtracted from $\mathbf{g}_{Aj}$ to arrive at lower bound $\mathbf{g}_{lAj}$ of the constraint and added to arrive at the upper bound $\mathbf{g}_{uAj}$.

Since the Random Forest Regression models uses bootstrap samples, predictions using the samples left out can be used to estimate generalization error. It has been repreatedly shown that the error estimated using this method is quite accurate, and for regression forests appear to be even slightly pessimistic [24], [25]. These so-called out-of-bag estimates are used here to calculate R^2, where $R^2 = 1 - \frac{sum\_of\_squares_{residual}}{sum\_of\_squares_{total}}$.

The goodness of fit of the models is estimated for each of the constraint variables using adjusted R^2, where $R^2_{adj} = 1 - (1 - R^2)\frac{N-1}{N-K-1}$ with sample size N and number of predictors K. R^2_{adj} adjusts for the number of predictors that are added; unlike R^2, R^2_{adj} only increases if the added term increases the explained variance significantly. If the R^2_{adj} exceeds a threshold of 0.3, the predicted constraint boundaries are used. If R^2_{adj} falls below the threshold, it is assumed that the motion parameter does not vary predictably and a simple statistical measure is used. In that case $\mathbf{g}_{lAj}$ is set to equal the first quartile of that variable from all demonstrations, and $\mathbf{g}_{uAj}$ is set to equal the third quartile.

E. Performance evaluation

The acquired models are tested by evaluating the ability of the system to perform the pouring action. For this assessment the controllers perform pouring actions in simulations with random (previously unseen) container volumes and starting positions. The algorithm used to apply the controller to the scenario is specified in Algorithm 1.

Initial Position [x,y,z] (m)	Container Volume (ml)	Simulated Liquid (ml)	Poured Liquid (ml)	Spilled Liquid (ml)	phase1 max-velocity(m/s)	phase1 cup-bottom-above-maker(m)	...
(-0.11,0.26,0.039)	863.94	109.5	66.43	0	0.5772	0.2615	...
(-0.091,0.39,0.039)	725.71	109.5	90.52	0	0.4924	0.2006	...
(0.16,0.56,0.039)	587.48	109.5	61.32	0	0.9555	0.2394	...
(-0.17,0.67,0.039)	449.25	109.5	70.81	0	0.8843	0.2524	...
(0.17,0.68,0.039)	138.23	109.5	62.78	0	0.8773	0.1559	...

TABLE I

EXAMPLE TRAINING DATA

Algorithm 1 Evaluation of the learned action models for context-sensitively predicting pouring motions. We switch between motion phases after a predicted period of convergence has passed.

function EVALUATE($ActionModel, \Delta\omega, \Delta v$)
  $Sim.Init()$
  $inputs \leftarrow Sim.Cup.Pose \cup Sim.Maker.Pose \cup Sim.Cup.Fill$
  $Motions \leftarrow Predict(ActionModel, inputs)$
  for m **in** $Motions$ **do**
    $c \leftarrow CreateController(m.model)$
    $Buf \leftarrow \{\}$
    while not $Converged(Buf, m.Wait, \Delta\omega, \Delta v)$ **do**
      $Buf \leftarrow Buf \cup Sim.Cup.Twist$
      $obs \leftarrow Sim.Cup.Pose \cup Sim.Maker.Pose$
      $Sim.Cup.DesTwist \leftarrow c.Compute(obs)$
      $Sim.Step(1ms)$
    end while
  end for
end function

function CONVERGED($Buf, WaitSamples, \Delta\omega, \Delta v$)
  if $Buf.Size < WaitSamples$ **then**
    return False
  end if
  for $i \leftarrow Buf.Size - WaitSamples$ **to** $Buf.Size$ **do**
    $t \leftarrow Buf[i]$
    $\omega_{max} \leftarrow max(\{|max(t.rot)|\} \cup \{|min(t.rot)|\})$
    $v_{max} \leftarrow max(\{|max(t.trans)|\} \cup \{|min(t.trans)|\})$
    if $\omega_{max} > |\Delta\omega|$ **or** $v_{max} > |\Delta v|$ **then**
      return False
    end if
  end for
  return True
end function

The system was tested in a learned condition and a fixed condition. In the former, variables were predicted as described in Section IV-D. In the latter, all constraints were estimated from the demonstrations using the static interquartile range. These two conditions are compared to the human demonstrations to assess how well the controllers perform. They are also compared to each other to test the hypothesis that the model has merit over a statically defined controller. Each condition was tested through 200 simulations.

We use the same recording and segmentation procedures for the actions performed by our system as we used for the human demonstrations. The following dimensions are considered important for evaluating this pouring task: whether a pouring action was performed, whether spilling occurred, how much was spilled, and the size of the resulting pancake.

V. RESULTS

The results are shown in Table II. A total of 211 pouring demonstrations, of which 187 without spilling, were collected during the experiment. There were approximately the same amount of complete, non-spilling demonstrations for each container volume condition, the number of demonstrations ranging from 29 to 33.

For the Controllers (learned) condition, 196 episodes contained successfully detected pouring actions, two of which contained spillage. The amount of successful pouring actions without spill is higher than of the human demonstrators (97% and 88.63% respectively). Four episodes failed because liquid left the cup after tilting back had completed, thus mismatching the expected event pattern.

For the Controllers (fixed) condition, 176 episodes contained successfully detected pouring actions, three of which contained spillage (86.5% success). Of the 26 that did not contain pouring, 3 had the same reason for failure as those in the learned condition. The remaining 23 episodes did not contain pouring because the cup was not tilted enough for liquid to leave the cup.

The fixed controllers have an average of amount poured that is closer to the target than the learned controllers. This comes at the cost of not being able to do the task appropriately when the cup is either quite full (spilling) or quite empty (no liquid poured) however, resulting in 20 less succesful pouring actions than the learned controllers.

Noticably, in the episodes where pancake mix was spilled, the amount spilled by the controller (0.73ml on average) was considerably less than the human demonstrators (9ml on average). The demonstrators were much more accurate in how much was poured onto the pancakemaker however, overshooting the target amount on average by only 3.49ml.

The histograms in Figure 3 show that whereas the human demonstrators demonstrate a nice normal curve around the target volume, the fixed controllers display a distribution all over the place. The learned controllers' curve is similar to the demonstrators', but with a bias towards pouring too much. There are many possible reasons why such a bias might exist. These are likely to be specific to this action however (for example how the motion descriptions were constructed). Importantly, it suggests that a mechanism for monitoring and correcting such behavior is desired in the framework.

In terms of time it took to execute the task, the humans were considerably faster than either controllers and showed more variance. The learned controllers were faster than the fixed controllers and showed more variance as well.

Actor	Episodes (Eps) Overall	Eps. with Pouring	Eps. with Spilling	Mean Spilled Volume for Eps. with Spillage (ml)	Mean Poured Volume for Eps. with Pouring (ml)	Mean Difference to Targeted Volume (ml)	Mean (SD) Task Time (sec)
Subjects	211	211	24	9.00	58.24	+3.49	7.57 (2.77)
Controllers (learned)	200	196	2	0.73	88.93	+34.19	10.15 (1.83)
Controllers (fixed)	200	176	3	0.73	60.45	+5.70	11.05 (1.38)

TABLE II

SUMMARY OF THE POURING EPISODES COLLECTED

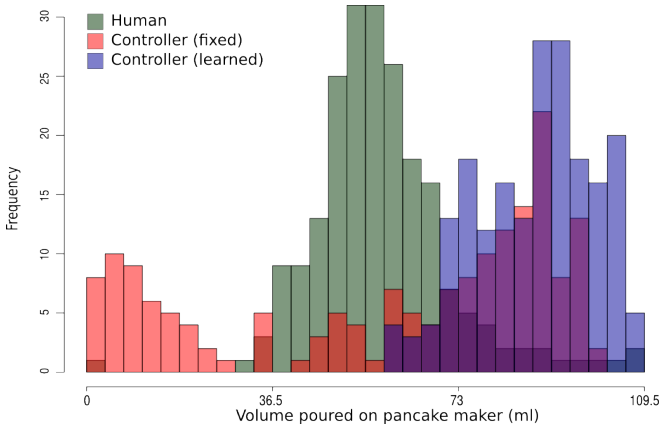

Fig. 3. Histograms of volume poured onto the pancake maker

Motion phase	Constraint variable	R^2_{adj}
Move above pancakemaker	cup-top-above-bottom	0.7954
Tilt container down	cup-top-above-bottom	0.8437

TABLE III

GOODNESS OF FIT FOR THE MODELS THAT EXCEEDED THE THRESHOLD.

An overview of the constraint variables whose R^2_{adj} score exceeded the threshold is given in Table III. The constraint variable "cup-top-above-bottom" is the difference in z between the top and the bottom of the cup. It is therefore a measure of tilt in the z-axis. For all phases, this variable is the most reliably predicted from the environment variables. This is to be expected since the largest source of variation for the action in the current setup is the fullness of the container, and this is expected to have a great influence on how much the cup is tilted during lift and pouring.

VI. CONCLUSION AND DISCUSSION

In this paper, a framework for acquiring action models was proposed that combines a flexible control framework with a generally applicable learning framework. The system was able to perform context-sensitive actions reliably by learning from non-experts in a simulated environment.

[26] have a similar approach to collecting demonstrations in simulation from non-experts using an intuitive interface to teach robots actions in a virtual environment. They collect information to decide which tasks to perform, whereas our approach emphasizes the importance of learning *how* actions should be performed.

Enabling non-experts to teach robots, often refered to as *Learning by Demonstration*, has received considerable attention as a way to acquire action knowledge [27]–[29]. The approach is interesting because it would enable robots to learn (about) an enormous amount of actions at low cost. Moreover, it can enables systems to learn important aspects of tasks that are difficult for humans to specify explicitly or which humans themselves might be unaware of.

By using demonstrations rather than a large set of automatically generated simulations, we are ignoring a large proportion of the possible motion space. While taking some "human time", it drastically reduces the required computation time. We believe that having humans highlight a path through the vast search space will prove a valuable strategy for constructing a general framework for learning to perform a wide range of everyday manipulation tasks reliably.

Two key questions that arise in LfD is what to imitate and how. The first question addresses which features are relevant for reproducing an action. E.g. what describes the essence of an action? There appear to be keypoints during actions that define the action type. Flanagan *et al.* [30] show that changes in contact events are of great importance for signaling the transition between phases of an action, a phase for example being grasping or moving an object. This view is supported by the findings of Aksoy *et al.* [18], who show that it is possible to recognize actions based solely on the sequence of changes in object relations. Here we used a simple version of this framework to divide actions into phases, using physical events to recognize and parameterize the action of interest.

The system is limited in that it does not monitor the events online. Thus, if the cup is not tilted enough and no pouring occurs, this is not detected until after the episode has completed. Adapting the system such that it monitors the ongoing process and motion phases and goals are triggered by expected events would likely enable it to perform better. Such a system is also desirable in order to be able to abort an action prematurely if unexpected events happen; for example, to stop pouring as soon as a spill is detected.

The second question addresses how the demonstrations should be used to generate feasible trajectories for the robots manipulators. The system presented here parametrizes the goals of flexible controllers instead of imitating demonstrated trajectories. Consequently, the learning module benefits from all desirable properties engineered into the control framework, e.g. motion smoothness or robustness to small disturbances. The controller parametrization is done through matching the design of the learning module to the controllers. To do so, we have given prior knowledge in the structure of the task functions by specifying motion phases and

variables. We intend to address this issue by for example enabling the controller to flow from one motion phase to another through automatically defined segments and goals taken from the demonstrations. Here we first demonstrate the feasibility of using such a control and learning framework for parameterizing context-sensitive actions.

Using physics simulations as environments to acquire action models from demonstrations is an attractive proposition. The resulting well-controlled and fully observable experiments promise to yield learning data of high quality and relevance. On the other hand, it may be hard to obtain accurate values for some of the necessary object parameters for new task domains. We believe that employing default values as best guesses will be sufficient to obtain qualitatively correct action models which enable robots to competently but not optimally perform and reason about new actions.

The framework can be further tested by adding more variation to the task such as changing the viscosity of the liquid, handling containers of various shapes and openings, varying the target size, etc. Extending the scenario to more input variables from the environment should be straightforward. Since the actions performed by the motion controllers are analyzed in the same fashion as the human demonstrations, this data can be fed back into the system to expand the training set, with the human demonstrations as starting points in this high-dimensional space.

In conclusion, the results show that the controller and learner can be combined meaningfully and capture enough general knowledge from the demonstrations to perform the actions in other instances. Though the currently presented framework has its limitations, it shows promising results in learning general, context-sensitive action models.

ACKNOWLEDGMENT

The authors thank Andrei Haidu for his help with the simulation environment setup and recording.

REFERENCES

[1] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011, pp. 1470–1477.

[2] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *Int. Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.

[3] M. Tenorth and M. Beetz, "KnowRob – A Knowledge Processing Infrastructure for Cognition-enabled Robots," *Int. Journal of Robotics Research*, vol. 32, no. 5, pp. 566 – 590, April 2013.

[4] ——, "Representations for robot knowledge in the knowrob framework," *Artificial Intelligence*, 2015.

[5] S. Calinon and A. Billard, "A probabilistic programming by demonstration framework handling constraints in joint space and task space," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, Nice, France, September 2008, pp. 367–372.

[6] L. Kunze, M. E. Dolha, E. Guzman, and M. Beetz, "Simulation-based temporal projection of everyday robot object manipulation," in *10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, Taipei, Taiwan, May, 2–6 2011.

[7] L. Kunze, A. Haidu, and M. Beetz, "Acquiring task models for imitation learning through games with a purpose," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2013, pp. 102–107.

[8] A. Haidu, D. Kohlsdorf, and M. Beetz, "Learning task outcome prediction for robot control from interactive environments," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Chicago, USA, 2014.

[9] E. Aertbeliën and J. De Schutter, "eTaSL/eTC: A constraint-based task specification language and robot controller using expression graphs," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2014, pp. 1540–1546.

[10] M. Tenorth, G. Bartels, and M. Beetz, "Knowledge-based specification of robot motions," in *Proc. of the European Conference on Artificial Intelligence (ECAI)*, 2014.

[11] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, vol. 3, 2004, pp. 2149–2154.

[12] A. Haidu, D. Kohlsdorf, and M. Beetz, "Learning action failure models from interactive physics-based simulations," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, 2015.

[13] G. Ye and R. Alterovitz, "Demonstration-guided motion planning," in *Int. Symposium on Robotics Research*, 2011.

[14] D. Kulić, C. Ott, D. Lee, J. Ishikawa, and Y. Nakamura, "Incremental learning of full body motion primitives and their sequencing through human motion observation," *Int. Journal of Robotics Research*, vol. 31, no. 3, pp. 330–345, Mar. 2012.

[15] G. Konidaris, S. Kuindersma, A. G. Barto, and R. A. Grupen, "Constructing skill trees for reinforcement learning agents from demonstration trajectories." in *Advances in Neural Information Processing Systems*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds., 2010, pp. 1162–1170.

[16] E. Fox, M. Hughes, E. Sudderth, and M. Jordan, "Joint modeling of multiple related time series via the beta process with application to motion capture segmentation," *Annals of Applied Statistics*, vol. 8, no. 3, pp. 1281–1313, 2014.

[17] D. Kulic, W. Takano, and Y. Nakamura, "Combining automated on-line segmentation and incremental clustering for whole body motions." in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2008, pp. 2591–2598.

[18] E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter, "Learning the semantics of objectaction relations by observation," *Int. Journal of Robotics Research*, vol. 30, no. 10, pp. 1229–1249, Sept. 2011.

[19] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, oct 2001.

[20] P. Latinne, O. Debeir, and C. Decaestecker, "Limiting the number of trees in random forests," in *Multiple Classifier Systems*, vol. 2096, 2001, pp. 178–187.

[21] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas, "How many trees in a random forest?" in *Proc. of the 8th Int. Conf. on Machine Learning and Data Mining in Pattern Recognition*. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 154–168.

[22] R. A. Bailey, *Design of comparative experiments*. Cambridge University Press, 2008, vol. 25.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[24] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R News*, vol. 2, no. 3, pp. 18–22, 2002.

[25] L. Breiman, "Out-of-bag estimation," UC Berkeley, Department of statistics," Technical Report, 1996.

[26] N. P. Koenig and M. J. Mataric, "Training wheels for the robot: Learning from demonstration using simulation." in *AAAI Fall Symposium: Robots Learning Interactively from Human Teachers*, ser. AAAI Technical Report, vol. FS-12-07. AAAI, 2012.

[27] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, May 2009.

[28] S. Ekvall and D. Kragic, "Learning Task Models from Multiple Human Demonstrations," in *15th IEEE Int. Symposium on Robot and Human Interactive Communication*, sep 2006, pp. 358–363.

[29] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot Programming by Demonstration," in *Springer Handbook of Robotics*, 2008, pp. 1371–1394.

[30] J. R. Flanagan, M. C. Bowman, and R. S. Johansson, "Control strategies in object manipulation tasks," *Curr. Opin. Neurobiol.*, vol. 16, no. 6, pp. 650–659, Dec 2006.