Learning Task Outcome Prediction for Robot Control from Interactive Environments

Andrei Haidu[1] ahaidu@cs.uni-bremen.de
Daniel Kohlsdorf[2] dkohlsdorf6@gatech.edu
Michael Beetz[1] beetz@cs.uni-bremen.de

***Abstract*— In order to manage complex tasks such as cooking, future robots need to be action-aware and posses common sense knowledge. For example flipping a pancake requires a robot to know that a spatula has to be under a pancake in order to succeed. We present a novel approach for the extraction and learning of action and common sense knowledge, and developed a game using a robot-simulator with realistic physics for data acquisition. The game environment is a virtual kitchen, in which a user has to create a pancake by pouring pancake-mix on an oven and flipping it using a spatula. The interaction is done by controlling a virtual robot hand with a 3D input sensor. We incorporate a realistic fluid simulation in order to gather appropriate data of the pouring action. Furthermore, we present a task outcome prediction algorithm for this specific system and show how to learn a failure model for the pouring and flipping action.**

I. Introduction

In order for future robots to successfully accomplish more and more complex manipulation tasks, they are required to be action-aware. They need to possess a model which discloses how the effects of their actions depend on the way they are executed. For example, a robot making pancakes should have an understanding of the effects of its actions: pouring the pancake-mix on the oven depends on the position and the way the container is held, or that the consequence of sliding a spatula under a pancake may or may not cause damage to it, depends on the angle and the dynamics of the way it is pushed. In artificial intelligence these are considered to be naive physics reasoning capabilities.

Robots having a naive physics understanding of their manipulation actions are able to perform considerably better. A robot, knowing that the outcome of pouring depends on the height of the container and the relative holding position to the center of the oven, can learn pouring skills much more efficiently. Such knowledge can guide exploration in reinforcement learning. Or it can focus imitation learning on aspects of the demonstration that are known to change the action effects and thereby decide what to imitate and what not. Models of different expected aspects and effects of certain tasks, can also be used to monitor task execution. Furthermore, the robot can respond to perceptual inputs in a better informed way.

Unfortunately naive physics and commonsense knowledge is applied unconsciously and is seldomly communicated since it is assumed that everybody possesses it. The result is that these forms of awareness are difficult to state explicitly.

Fig. 1. Game Setup

Kunze et al.[8] have proposed games with a purpose as means to equip robots with commonsense and naive physics knowledge. The basic idea is to design computer games that include virtual scenes in which virtual manipulation tasks are to be performed.

Our current gaming setup is depicted in Figure 1. The system is provided with a sensor infrastructure that allows interaction with the virtual world by tracking the player's hand motion and gestures and mapping them onto the robotic hand in the game. We tested out two different setups for the tracking. One with the help of a magnetic sensor based controller, which returns the position and orientation of the hand, together with a dataglove for finger joints positioning. In the second case we used two 3D camera sensors mounted on a frame which yields the pose and the skeleton of the tracked hand.

This paper goes beyond the work of Kunze et al. by supplementing the rigid body simulator with a realistic fluid model, thus being able to gather more accurate data when manipulating fluids. Improving the handling of the game, by making the objects easier to manipulate. For example, by adding dampened joints to the spatula parts in order to mimic bending when it is pushed against the surface of the oven. Furthermore, if such a system is used more widely by a variety of users, we can use the resulting data to learn models of task performance and outcome. In that way we have a

[1] are with the Institute for Artificial Intelligence and the TZI (Center for Computing Technologies), Universität Bremen, Germany.
[2] is with the Contextual Computing Group, Georgia Institute of Technology, Atlanta, USA.

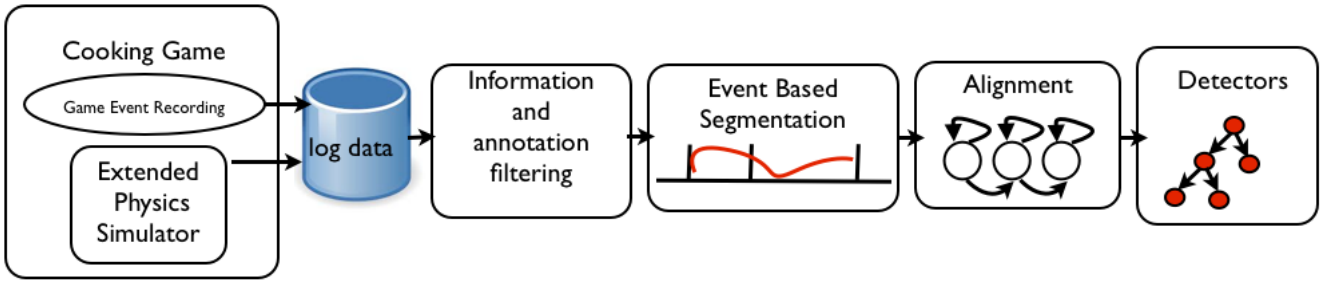

Fig. 2. System Overview

basis for learning important aspects of a task without stating action and common sense knowledge explicitly. In particular we use the hand pose and position trajectory from multiple gaming sessions in order to build a failure detection system. The goal is to learn a model that can predict early when a task is about to fail, based on the current situation. Such failures can include wrong angles of the spatula or missing the oven while pouring. Using such a failure detection system a robot might notice that its current action could lead to a failure and decide to change its current plan of action accordingly.

In the remainder of the paper we proceed as follows. In the next section we give a functionality overview of the system as a whole. In Section III we describe in greater detail the simulated environment, the various interaction methods with the game and the augmented fluid model. In section IV we illustrate how we build a task outcome predictor using data from the gaming environment and present our empirical findings in section V. We introduce and discuss related work in Section VI and finally conclude and present future work in Section VII.

II. Overview

The system, is depicted in Figure 2 and it consists of two main parts. The first part is the cooking game, designed in a robot-simulator. The simulator uses a physics engine capable of simulating precise real-time rigid body dynamics, which is extended with the ability to simulate realistic fluids. During simulation physics data and interpreted game events are automatically recorded. The second part is a task outcome prediction system that uses the data recorded from the game. We frame the goal of the task outcome prediction system as a binary classification problem, mapping event data together with the game's current state to a prediction for success or failure. During learning and prediction we use event-based segmentation to structure the game into episodes such as "taking the pancake-mix and pouring it on the oven" or "picking up the spatula and flipping the pancake" and learn isolated prediction models for each of these episodes.

The "cooking game" consists of a virtual kitchen equipped with an oven, a spatula and a container filled with the pancake-mix fluid. The player's task is to successfully make a pancake with the help of a simulated anthropomorphic robotic hand. The actions involved are grasping the container filled with the mix, carrying it above the oven and pouring the right amount of liquid on it in a way that it forms a normal sized pancake. Eventually, using the spatula, the player needs to slide it under the pancake, lift and turn it in order to make the pancake flip it on its opposite side to 'cook' evenly.

The different manipulation actions can be performed in numerous approaches, meaning the successes or failures, and the related effects critically depend on the way they are executed. For instance, the pouring effects strongly rely upon the various heights, angles, and tilting speeds of the container, which can yield different pancake forms and sizes. Flipping the pancake, again, depends on the chosen height, speed and rotation axis of the spatula head. In case of failures, such as spilling the mix, pushing the pancake off the oven, unsuccessful flipping, or other undesirable effects, the recorded data is labeled with the failure event. Thus, by observing game episodes the system can accumulate information about successes and failures, giving the robot a faster way to predict the outcome of its action.

In order to learn the task failure detector, we extract the positions and orientations of the robotic hand, as well as the first and second derivative as features from the logged data. Furthermore, we segment the data into episodes using specific events from the simulation such as "take spatula" or "drop spatula". For example, if we were to detect an error during the flipping action we would only consider the segment in which the user holds the spatula. During learning we use Hidden Markov Models to align episode segments to a discrete set of states. In that way we achieve a temporal grouping of all frames (pose of hand) in a segment. We then learn multiple frame level failure detectors using the data of the states where failure and success outcomes are highly separable. In that way we can decide for every frame if the current action will lead to success or failure.

III. Simulation Environment

In the following section we describe in detail the game framework with the types of user interaction and the extension of the physics engine with a fluid model.

A. The Game Framework

The robot-simulator used to create the cooking game environment is Gazebo [7], a multi-robot simulator capable of achieving physically-plausible object interactions using ODE [16] as its physics engine. Its main components include

the rigid body dynamics and the collision detection engine. In [4], drawbacks such as poor robustness solver, lack of computational efficiency, weak support for joint-dampening and friction approximation by linearization were pointed out, and solved by creating extensions which enabled multi-threaded execution, added viscous joint dampening and a convex-optimization solver. With these additions, the computational speed and precision of the simulation has greatly improved, allowing the possibility of running complex scenarios in real-time, such as our cooking game environment.

The interaction between the player and the virtual environment has been done by tracking the user's hand motions and gestures and mapping them in real-time onto a simulated anthropomorphic robotic hand, namely the DLR/HIT four finger robotic hand model. For the hand tracking we tested out two different systems. Firstly, we used Razer Hydra's motion sensing game controller, which uses a weak magnetic field in order to compute the position and orientation of its sensor with a precision of up to 1 mm and 1 degree. For finger tracking we used the X-IST dataglove, a motion capture device which uses bend sensors for each finger joint to calculate their position. To combine the two devices we separated the internal sensor from the controller and attached it to the glove at the wrist position. Using this approach gives the advantage of consistency and precision in tracking the hand, however the high price of the dataglove prevents the possibility of eventually crowdsourcing the game.

To cope with the crowdsourcing idea the second system tested was done with the 3Gear systems framework, which enables Kinect-based cameras to reconstruct movements and gestures of a human hand. The advantages in this case was the ability to use a popular and affordable sensing device where the player did not need any additional hardware to track his hand. However, in some positions the hand rotation could not be properly tracked, an action critical for the pouring and flipping task. This led us to currently run our simulation and gather data using the first alternative.

The control of the virtual robotic hand is implemented using a PI controller, which computes every timestamp the required forces and torques needed to be applied on the wrist in order to move the hand correspondingly to the player's one. The communication between the sensors and the game framework is done via ROS [14] messages.

B. Realistic Fluid Model

One of the main disadvantage in modeling a liquid-like material using exclusively rigid body dynamics, is the amount of tweaking required to be done to the physics engine in order to simulate a model that behaves roughly as a liquid. Prior to the extension of the physics engine with the fluid model, we managed to do this by using numerous solid spheres acting as liquid particles. To avoid strong elastic collisions between them, we set the material characteristics of the spheres to a very soft one. Meaning, in case of collision, spheres can penetrate each other by greatly dampening the repulsive force acting between them. Another obstacle was the lack of rolling friction in ODE, to cope with it, we created a parameterizable controller for each sphere, which dampened their angular velocity when in contact with the surface of the oven. Thus giving us a naive control on the viscosity of the 'liquid' after it has been poured. The major disadvantage of using this approach was the inability of running the simulation in real-time when increasing the amount of liquid particles (>40).

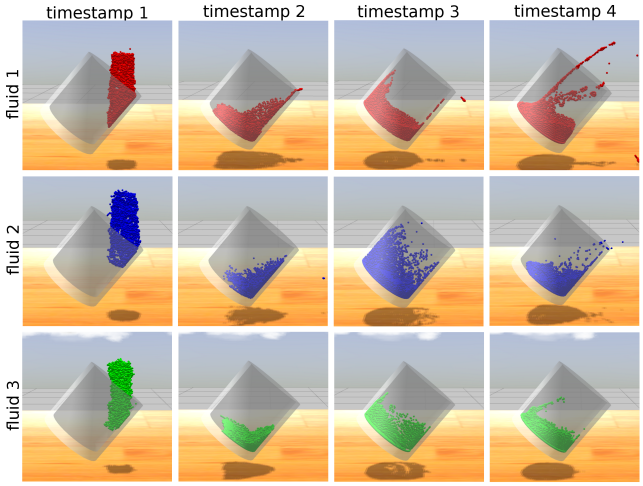

Fig. 3. Liquid examples

To resolve this issue we implemented a realistic fluid model based on Monaghan's [11] Smoothed Particle Hydrodynamics (SPH) method. The fluid is composed of multiple interacting particles, making it an embarrassingly parallel problem. To take advantage of this we chose an implementation which enables the computation to be executed on the GPU as well. The framework we used is Fluidix [9], a CUDA-powered particle simulation library, providing a fully dynamic support for interactions between particles and external objects. By running the parallel code on the computers graphical unit we had the possibility to simulate thousands of particles without interfering with the real-time factor of the rigid body physics engine.

In Figure 3 we illustrate the interaction of the resulting implemented fluid when poured in a tilted, static cup. We used three types of fluids with identical particle count, but different parameters, such as mass-density, stiffness, viscosity, surface tension. In the middle row, fluid number 2, with blue colored particles, has the standard parameters of water, such as mass-density of $998.29\frac{kg}{m^3}$, viscosity of $3.5Pa \cdot s$, surface tension $0.0728\frac{N}{m}$. The top fluid, number 1, colored red, has the viscosity parameter increased compared to the water, and the lower one, number 3, represented with color green, has the characteristics of a jelly-like liquid. Having the simulated model created with formulas which take into account all the typical parameters of a real fluid, we have the possibility to accurately reconstruct different liquid types, which may be required in other simulation setups.

The SPH method we implemented is based on Kelager's work [6], where the interacting forces between the particles are derived from the Navier-Stokes equations for a weakly

compressible and isothermal fluid. The resulting Lagrangian formulation, yields

$$\rho \frac{du}{dt} = -\nabla p + \mu \nabla^2 u + f. \tag{1}$$

where, ρ is the mass-density, u the velocity, and on the right hand side, $-\nabla p$ represents the pressure term, $\mu \nabla^2 u$ the viscosity, and f the external forces acting on the particles, such as gravity.

SPH is basically an interpolation method using the concept of integral representation of any quantity function $A_I(r)$, at a point r in the field, with kernels that approximate Dirac's delta function:

$$A_I(r) = \int_\omega A(r')W(r - r', h)dr', \tag{2}$$

where ω is the domain, W is a smoothing kernel with the support radius of h. The equation is then discretized by replacing the integral operation with a summation, and the differential element dr' with the volume V which in turn is equal to the particle mass m_j divided by the mass-density ρ, yielding

$$A_S(r) = \sum_j A_j \frac{m_j}{\rho_j} W(r - r_j, h) \tag{3}$$

The Gradient ∇ and Laplacian ∇^2 of the discretized quantity function $A_S(r)$ can be calculated by taking the first and second derivative of the smoothing kernel W.

To determine the mass-density acting on a specific particle i, we use the general SPH formulation (3) to compute the field. At particle i the mass-density yields

$$\rho_i = \sum_j m_j W(r_i - r_j, h). \tag{4}$$

The pressure, an internal force emerging only from interactions between the particles, is the main cause for the repulsive and attractive forces between neighboring particles. The pressure p at a given particle can be computed with the ideal gas law equation. However, using it will result in purely repulsive forces, because of the pressure field which will always be positive. Desbrun in [3] introduces a modified version of the ideal gas equation, using an additional rest density ρ_0 to overcome this situation, yielding the pressure to $p = k(\rho - \rho_0)$, where k is the gas stiffness constant. From the pressure field, we now can compute the pressure force at particle i, by using the gradient of the standard formulation (3) where we replace $A_S(r)$ with the pressure term $-\nabla p$ from equation (1), resulting

$$-\nabla p(r_i) = -\sum_{j \neq i} p_j \frac{m_j}{\rho_j} \nabla W(r_i - r_j, h). \tag{5}$$

The viscosity term arises from interaction within the particles and defines the resistance to flow. The SPH variant can be computed by introducing it in the standard interpolation scheme (3), which yields,

$$\mu \nabla^2 u(r_i) = \mu \sum_{j \neq i} u_j \frac{m_j}{\rho_j} \nabla^2 W(r_i - r_j, h). \tag{6}$$

The smoothing kernels functions (W) for default, the pressure and the viscosity were used in the form suggested by Müller in [12].

The surface tension is applied as an external force, which acts on the exterior of a liquid fluid. It is not present in the Navier-Stokes equations since it only effects the fluid at its boundaries. The force induces the particles to shift towards the liquid, in the direction of the surface normal. The effect tends to minimize the liquid area by flattening the surface curvature.

A more comprehensive explanation of SPH, its implementation and the underlying formulas merely omitted here, can be found in the work of Kelager [6].

IV. Task Outcome Prediction

In the following we will describe how to learn a task outcome predictor. The goal is to learn a model that can decide if the current action observed on a frame level will lead to a success or failure of the task.

We record our data as trajectories from our game simulation. We label each trajectory with success or failure of the task at the end of the simulation. A trajectory labeled with success represents pouring the pancake-mix and flipping the pancake successfully. On the other hand, a trajectory labeled with failure might occur under different conditions. For example flipping the pancake off the oven. In the current implementation we extract the hands position in x, y, z from the simulation as well as its orientation in angles as $\alpha_x, \alpha_y, \alpha_z$ at a constant frame rate as our features. We use the data from 26 successes and 19 flipping off trials.

Our task outcome prediction system is divided into three stages: Segmentation, Data Preprocessing, Learning and Prediction. In the first step we segment the data using semantic events. Semantic events provide context such as "we grab the spatula" or "we touch the oven". These events divide a trajectory into meaningful segments. For example the event "grabbing the spatula" together with "dropping the spatula" will cut out the segment in which we flip the pancake. In the data pre-processing step we align a set of example segment trajectories to a Hidden Markov Model. In that way we can gather samples for each state in isolation for learning. From the samples of states that seem promising for good classification into success and failures we learn multiple classifiers. During outcome prediction we classify every new sample using all of the classifiers in the appropriate segment and use a weighted cumulative count to detect failure as early as possible.

A. Data Preprocessing and Learning

Since we use human subjects to perform the tasks the execution timing will change between subjects and also between trials. Using an alignment first, we account for this so called "time warp" and force each samples of each trajectory into one state. Since we assume sequential actions we use

a left to right Hidden Markov Model topology, meaning we can only stay in a state or transition to its successor. One can see this alignment step as a temporal clustering into the states of the HMM. That means that similar frames will be associated automatically with a particular state. The first step of the data preprocessing is to align the trajectories against a common Hidden Markov Model. Therefore, we estimate its parameters using multiple iterations of Baum Welch [15] using only examples labeled success. We compute the most likely state sequence for all examples (success and failure) using the Viterbi algorithm [15]. One can see the estimated model as one for the normal non failure case. By aligning failure and successes against it, we search for states were large deviations from the normal case occur. These alignments are used to visually inspect the samples in each state. We create a scatter plot with the states on the x-axis and one of the dimensions on the y-axis. Such a plot for our data is shown in Figure 4. The color of the sample codes successes and failures.

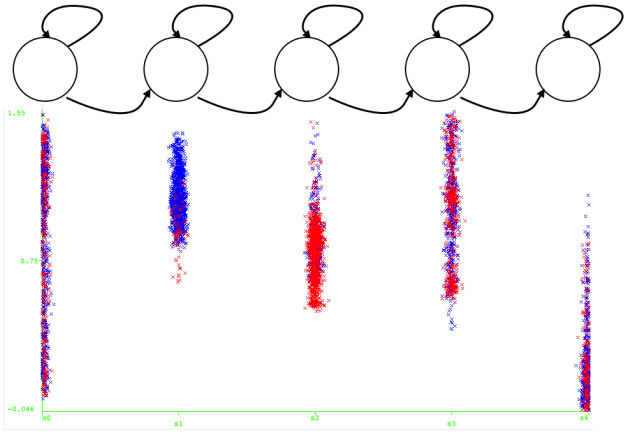

Fig. 4. We use a 5 state HMM. Under each state we plot all samples in the α_z dimension. Again we plot all failures in red and all successes in blue. As one can see, state two and three seem promising for classification. Furthermore, these states are early enough to allow fast failure feedback.

Using this plot we search for early states with "high" scattering between the two classes. In that way we expect the failure detectors to trigger early in the case of a failure and the errors can be detected with high accuracy. After choosing a promising state, we proceed with learning a failure detector on the error samples in that state. In order to account for false triggering we construct the training set such that we train the failure samples of that state against all other success samples in that segment. The resulting training set will consist of way more success examples. The failure class will be well under-represented. We account for the under representation by weighting the data set. A byproduct of the Baum Welch algorithm is the marginal $\gamma_s(t)$ probability for each sample at time t to each state s[15]. Assuming we choose a state s, we choose the weight for sample t as $\gamma_s(t)$, the probability of sample t belonging to state s. We repeat the above procedure for multiple states in order to cover different errors or deviations from the normal successful behavior. The result of the complete procedure is a set of binary classifiers voting for success or failure, trained from different error sources in chosen states.

B. Building the Predictor

The final component of our system is the actual task outcome predictor. Our goal is to use the binary detectors form the processing and learning component in order to decide when a particular state will lead to a failure during execution. Every sample in a trajectory is classified using all detectors. The drawback is that this will introduce a lot of false triggering, since we have no timing information at hand. One could align each time series against the Hidden Markov Model from the preprocessing and only classify in the state the classifier is trained for. However, using the Viterbi algorithm requires the time series to be known completely. In an online detection scenario this is not possible. But from the Hidden Markov Model we can calculate the expected number of samples we spend in each state and therefore the expected time we arrive in a certain state s. We calculated the expected number of samples for each state as the expected value of the geometric distribution defined by the probability of leaving a specific state:

$$\mathbb{E}(i) = \frac{1}{a_{ij}} \tag{7}$$

with a_{ij} being the probability of transitioning from state i to j. Now the expected number of samples up to that state is the sum of the expected samples for each preceding state:

$$\mathbb{E}_{upto}(s) = \sum^{s} \frac{1}{a_{ij}} \tag{8}$$

Since each classifier is estimated for a specific state we weigh each classification at time t by a Gaussian centered around $\mathbb{E}_{upto}(s)$. Given N classifiers $c_1 ... c_N$ trained on specific state $s_1 ... s_N$ and a threshold th we trigger a failure condition at time t when:

$$\sum_{i=1}^{N} c_i(t) * N(t|\mathbb{E}_{upto}(s_i), \sigma^2) > th \tag{9}$$

In other words we apply a threshold to a weighted sum of classifiers, while the weight is the likelihood of being close to the state the classifier was trained in.

In order to account for the performance of each classifier in isolation, we introduce another weight in the system representing the classification accuracy during performance acc_i. And the final detector is given by:

$$\frac{\sum_{i=1}^{N} acc_i c_i(t) * N(t|\mathbb{E}_{upto}(s_i), \sigma^2)}{\sum_{i=1}^{N} acc_i} > th \tag{10}$$

V. Experiments

We will describe our experimental setup and results for the different parts of the task outcome prediction system on data from the Gazebo pancake flipping scenario. As stated above we gathered 26 successful executions of the task and 19 failures by flipping off the pancake. For all the experiments we use a 5 state Hidden Markov Model for our data preprocessing algorithm. We estimated the threshold for the detectors empirically as 0.00003. Our implementation uses the Weka Toolkit for training the frame based detectors.

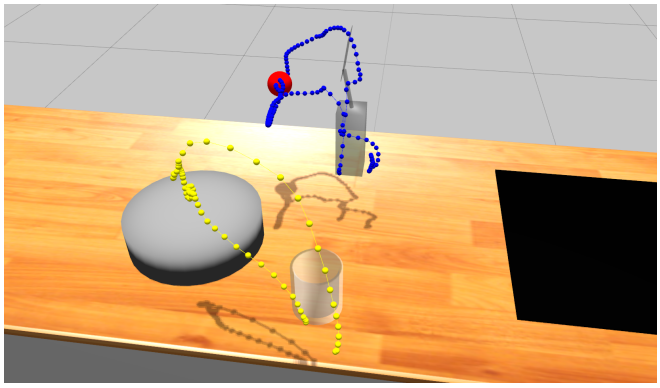

Fig. 5. Trajectory visualization with failure detection. The yellow trajectory shows a successful pouring. The blue trajectory shows flipping the pancake off the oven. The red marker shows the place where our detector found the error.

In our classification experiments we visualize the data as described above. From the visualization we decided to use data from state one, three and four. In order to get multiple votes for each error source we train multiple classifiers for each error source of the states. In order to evaluate which classifiers to use, we use the Weka toolkit and empirically try different classifiers on weighted data sets. We obtain the weights using the data preprocessing methods described above. In state one we found training a decision tree using the J48 algorithm to perform well and observe an accuracy of 98%. In state two we perform the same training for a decision tree (accuracy 99%) and a naive Bayes model (accuracy 96%). In state three we train a random forest with accuracy 99% and a decision tree again (accuracy 94%). In that way we have five detectors voting for success or failure. When applying our detectors to the trajectory data using the method described above we observe an accuracy of 71%. For our threshold we have six false positives and we miss seven errors. The confusion matrix is shown in Table I.

Given its accuracy our model is in no way capable of being the only source of success or error decision. However, we argue that it is a sufficient model to support such decisions. The two errors namely "false triggering" and "missing an error" can be resolved on other stages of a robot's motor control. The false triggering results in the robot replanning its action or backtracking. If these conditions do not happen very often, it will not effect the robot's performance. Missing an error can be handled when the robot is actually confronted with an invalid state.

	success	fail
success	20	6
fail	7	12

TABLE I

The results of our prediction experiments.

In our experiments we learned multiple decision trees using the J48 algorithm. This algorithm chooses node ordering using the information theory measure "Information Gain". In simple words, the attribute that will help the classification most, is picked first and will represent the root node. In our experiments the attributes we observed most in the higher levels were angles and velocities in position and posture. So the speed of the performance is important as well as the posture of the spatula. We think this indicates that the system is capable of learning simple physical reasoning capabilities. Choosing the wrong angle while sliding under a pancake will lead to a push off in real life as well, and performing the task too fast also. So the learned models meet our expectation in regard to the real world properties of the problem. In Figure 5 we show a successful pouring and an unsuccessful flipping event as well as the result of our detector. As one can see the error got detected early on when holding the spatula. Furthermore the spot marked is when the spatula touches the oven and the angle for the following action is decided. So the detector made the decision early based on the angle of the spatula.

VI. Related Work

Games with a purpose [18] have also been used for learning, however most of them used for labeling pictures over the internet, providing meaningful and accurate labels, or locating objects in them. In the work of [2] an approach using markerless tracking of human motion is presented. In our case we used a magnetic sensor attached to a dataglove or 3D cameras for hand tracking. Surgeons successfully managed to practice surgical procedures with the help of physics-based simulators and using haptic devices for controllers [10].

Hidden Markov Models are widely used in modeling sequence data such as speech [15]. Aligning multiple sequences against a Hidden Markov Model's states using multiple iterations of Baum Welch and Viterbi alignment is a standard procedure in Bio Informatics [5]. The goal of such an alignment is to group consecutive parts of a sequence. The Baum Welch algorithm will reestimate the observation distributions and the transition probabilities of a Hidden Markov Model such that the probability of the model generating provided training data is maximized. Given such a model we can use the Viterbi algorithm to find the most likely path through the Hidden Markov Model. That path provides an alignment against the states of the Hidden Markov Model. The frame level detectors we use are standard algorithms too. We chose "cheap" detectors such as decision trees, small random forests and naive Bayes in order to keep the workload of Task Outcome Prediction small. Furthermore these algorithms have shown to give good classification results in a variety of domains despite

their simplicity. Another reason for using decision trees in particular is that one can learn about the importance (in a information theory sense) of attributes in the data by looking at generated trees. For a deeper introduction to Decision Tree Learning we recommend reading any introduction to Machine Learning [13].

VII. Discussion and Conclusions

We presented a system to leverage naive physics knowledge and action models from computer games. Therefore, we developed a user controlled interactive environment that simulates creating of a pancake. The simulation supports fluid dynamics to build a realistic environment for data collection. Furthermore, we introduced a novel task outcome prediction algorithm using the temporal structure of our data. In our experiments we showed how event based segmentation, Hidden Markov Models and simple classifiers are not only capable of performing task outcome prediction but also showing meaningful detection.

In future work we are planning to implement an improved version of the fluid simulation by applying the Predictive-corrective incompressible SPH (PCISPH) method [17], which has the advantages of larger simulation time steps with low computational costs. Also adding new characteristics to the fluid, such as cohesion and friction between the particles [1], allowing us to simulate granular materials, such as salt or sugar. We are planning to improve the user interaction by trying out new systems and devices to track the hand and fingers of the player, also to output the visuals using stereo vision, in order to have a depth concept of the working environment.

References

[1] Iván Alduán and Miguel A. Otaduy. *SPH granular flow with friction and cohesion*. SCA '11. ACM, New York, NY, USA, 2011.

[2] Michael Beetz, Jan Bandouch, Dominik Jain, and Moritz Tenorth. Towards Automated Models of Activities of Daily Life. In *First International Symposium on Quality of Life Technology – Intelligent Systems for Better Living*, Pittsburgh, Pennsylvania USA, 2009.

[3] M. Desbrun and M.-P. Cani. A new paradigm for animating highly deformable bodies. *In Computer Animation and Simulation*, pages 61–76, 1996.

[4] E. Drumwright, J. Hsu, N. Koenig, and D. Shell. *Extending Open Dynamics Engine for Robotics Simulation*. Simulation, Modeling, and Programming for Autonomous Robots. Springer, 2010.

[5] R. Durbin. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[6] Micky Kelager. Lagrangian fluid dynamics using smoothed particle hydrodynamics, 2006.

[7] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. *In: Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2149–2154, 2004.

[8] L. Kunze, A. Haidu, and M. Beetz. Acquiring task models for imitation learning through games with a purpose. *International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[9] Adam MacDonald. Fluidix particle simulation api. *nVidia GPU Technology Conference*, 2013.

[10] A. Maciel, G. Sankaranarayanan, T. Halic, V. Arikatla, Z. Lu, and S. De. Surgical model-view-controller simulation software framework for local and collaborative applications. *International Journal of Computer Assisted Radiology and Surgery*, 2011.

[11] J.J. Monaghan. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, (30):543–574, 1992.

[12] M. Müller, D. Charypar, and M Gross. Particle-based fluid simulation for interactive applications. *Proceedings of 2003 ACM SIGGRAPH Symposium on Computer Animation*, pages 154–159, 2003.

[13] Kevin P Murphy. *Machine learning: a probabilistic perspective*. Cambridge, MA, 2012.

[14] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. Ros: an open-source robot operating system. *ICRA Workshop on Open Source Software*, 2009.

[15] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[16] Russel Smith. Ode: Open dynamics engine.

[17] B. Solenthaler and R. Pajarola. Predictive-corrective incompressible sph. *ACM Trans. Graph.*, 28(3):40:1–40:6, 2009.

[18] Luis von Ahn and Laura Dabbish. Designing games with a purpose. *Commun. ACM*, 51(8):58–67, August 2008.