Action Recognition and Interpretation from Virtual Demonstrations

Andrei Haidu
ahaidu@cs.uni-bremen.de

Michael Beetz
beetz@cs.uni-bremen.de

***Abstract*— To properly perform tasks based on abstract instructions, autonomous robots need refined reasoning skills in order to bridge the gap between the ambiguous descriptions and the comprehensive information needed to execute the implied actions. In this article, we present an automated knowledge acquisition system from human executed tasks in virtual environments, and extend the knowledge processing system KNOWROB[1] to be capable to reason on the acquired data. We have set up two scenarios in a physics based simulator: creating a pancake, and garnishing a pizza dough. Users where asked to execute these tasks using the provided tools and ingredients. Using a data processing module we then collect the low-level data and the relevant abstract events from the performed episodes. The recorded data is then made available in a format that robots can understand, by using a symbolic layer to interconnect the two data types in a seamless way.**

I. INTRODUCTION

Humans specify actions vaguely. They use instructions such as "add milk to the dough", where the person executing the instruction need to infer that the he has to pick up a container containing milk, hold it above the container with the dough, tilt it until the desired amount of milk is added, and so on. For an autonomous robot to able execute similar tasks from such abstract instructions, it needs to be able to infer this missing information. In Figure 1 we show an example of a robot reasoning on how to carry out the task of flipping a pancake. This kind of knowledge that every human posses is called commonsense and naive physics knowledge and it is needed by robots in order to carry out actions in open domains robustly and feasibly. Unfortunately this knowledge is difficult to state because it is implicit in the human cognition process. For example, people can show how to tie their shoes, but have great difficulty describing it.

In our research we aim at mining this knowledge by letting humans demonstrate such abstract instructions in an interactive physics based simulation using a *Game with a Purpose* (GwaP) [2]. Users would be asked to perform in a virtual reality environment various tasks described using abstract instructions. They would interact with the environment by mapping their natural hand movements onto a virtual end effector in order to manipulate objects, during which the system records the underlying motions and abstract events. Extracting the commonsense and naive physics knowledge requires the recognition of actions in a continuous stream of world evolution and segmenting them into phases. In this article we introduce an automated knowledge acquisition system from such environments and a KNOWROB extension to reason on recorded episodes.

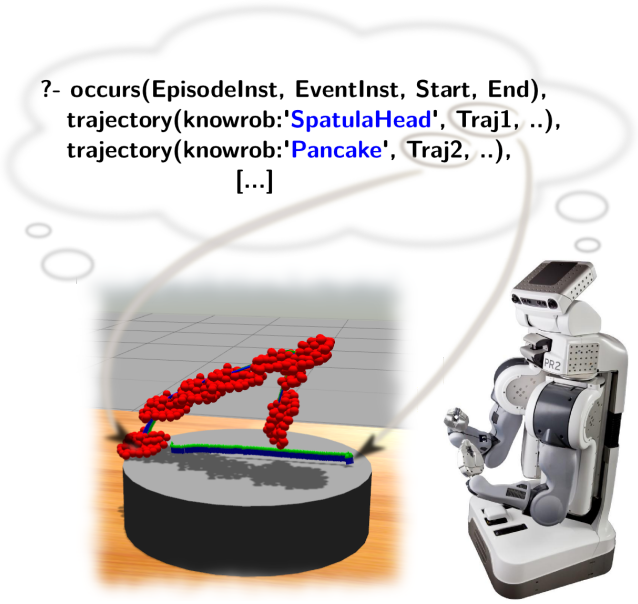


Fig. 1: PR2 querying the knowledge base for a successful pancake flip (query syntax presented in Section IV)

In the remainder of the paper we proceed as follows. In the following section we give a short overview of the system as a whole. In Section III we introduce the virtual environment setup and the representation technique of the recorded episodes in the knowledge base. Section IV describes various PROLOG queries for reasoning on the data collected from the two cooking experiments. Section V presents the related work. We then conclude and present our future work in Section VI.

II. OVERVIEW

The system architecture is comprised of two main parts: (1) the virtual environment based on a physics simulator with its data processing module and (2) the extended knowledge processing system KNOWROB. Figure 2 visualizes the aforementioned parts.

The first module of the system is the virtual environment. We set up two different scenarios to interact with. One is a pancake making world, where users are requested to create a pancake, this entailing to pour batter from a bottle on the pancake maker, and then flip the pancake on its 'un-cooked' side. A successful execution means that no batter shall be spilled during pouring and the pancake has to be flipped

The authors are with the Institute for Artificial Intelligence, Universität Bremen, Germany.

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

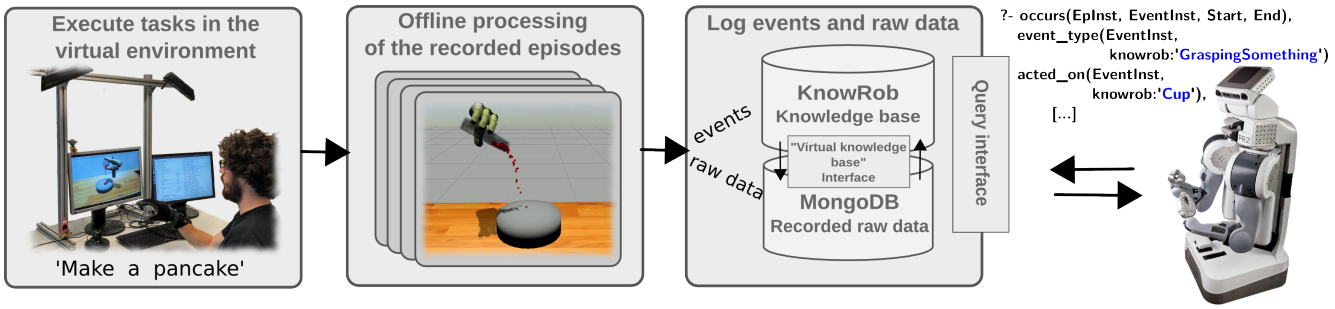


Fig. 2: System architecture

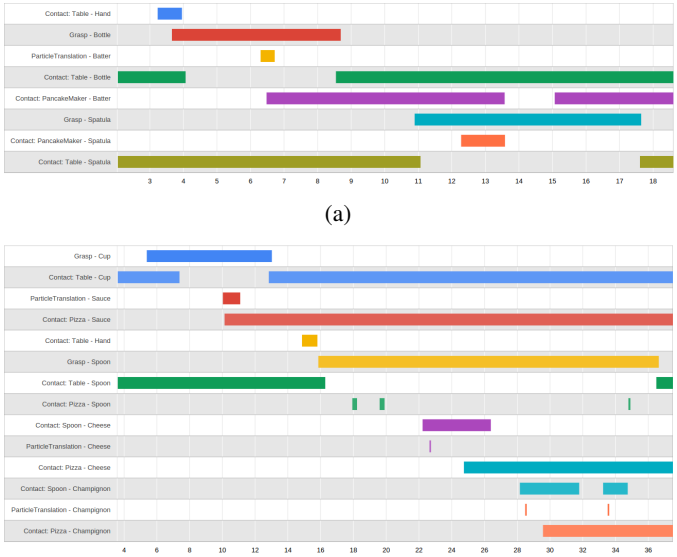
on its other side without falling off the pancake maker. The second world represents the pizza making scenario. The user is presented with a flattened dough, a container with tomato sauce, and two bowls with toppings. A successful execution entails pouring the sauce on the dough, spreading it using the spoon (in both cases spilling should be avoided), and scattering toppings from the bowls on the dough. For each scenario the users are presented with abstract instructions on what they have to do, for example, "make a pancake", "flip the pancake", "add cheese topping to the pizza" etc. For every recorded episode the manipulated objects were randomly arranged on the table surface.

Next, the data processing module re-plays the recorded episodes and extracts in parallel two types of information: (a) *raw data*, where for every simulation step the whole world state is logged. Involving the positions, orientations and bounding boxes of every model and their descendants. (b) relevant *abstract events*, by checking for specific contacts from the physics engine. Cognitive psychologists [3] propose models of actions that are specified in terms of force-dynamic states and events which are distinctive in the perceptual state. For instance, we detect a grasping event if an object is in contact with the fingers. Other possible detected events are particles leaving containers / falling off tools. We have visualized using timelines the specific events of a successful pancake and pizza making scenario in Figure 3.

After the episodes are stored in the knowledge base, the recorded data will be available in a format that robots can understand. A symbolic layer is created on the abstract concepts and the low-level data, connecting the two types in a seamless way. Thus, allowing robots to perform logical inferences without altering the original data structure. The technique is introduced in more detail in the following section.

III. REPRESENTATION OF THE EPISODES

The virtual environments were set up using the robotics simulator Gazebo [4]. The user interaction with the virtual world has been done by mapping the users' hand positions and orientations onto a simulated robotic hand. For hand tracking we used a motion sensing game controller, Razer Hydra, offering a precision of up to $1mm$ and $1deg$. In our previous work [5] we have showed that the collected data from the simulation, if set up properly, can be realistic



(a)

(b)

Fig. 3: (a) Pancake and (b) pizza making relevant events

enough to be applied in real life scenarios.

The representation of the recorded episodes relies on the action ontology of the robot knowledge processing system KNOWROB[1], which provides structures to represent actions, their spatial and temporal context, including events, objects, maps and robot components. It is implemented using PROLOG and can load knowledge stored in the Web Ontology Language OWL [6].

As introduced before, the processing module returns (a) the low-level data from the physics engine (stored in a large-volume database) and (b) the abstract events (stored in separate files using the aforementioned OWL statements). We use MongoDB[1] for storing the low-level data, each of its *database* represents different recorded scenarios (e.g. pancake / pizza making). These further consist of *collections*, representing the executed episodes. Following, a record in a collection is depicted by a *document* (JSON[2]-style data structure composed of field-and-value pairs), which stores all the relevant data for a given timestamp from the simulation

[1]http://www.mongodb.org/
[2]http://json.org/

episode. The recorded abstract events consist of instances of actions, events and assertions about the task context. For example the concept of grasping a bottle can be described as a subclass of *GraspingSomething* with the restriction that the *objectActedOn* has to be an instance of a *Bottle*. Each event instance also includes a *startTime* and an *endTime*, and using the systems built in methods we can reason about the time intervals using Allen's interval algebra [7].

To integrate a seamless communication between the abstract events, loaded in the knowledge base, and the separately stored raw data, a special feature of KNOWROB is used, a so called "virtual knowledge base". Conceptually and from the point of view of the queries, the low-level data appears as the rest of the stored information in the knowledge base. However, instead of storing such a large amount of raw information in a symbolic form, it is accessed only when requested, at query time. The advantages brought on by this approach are: the raw data does not need to be loaded in the knowledge base, hence the queries will be significantly faster, and the possibility and store the data using optimized databases, and process only the data that is required for the query.

IV. EXPERIMENTS

Having multiple episodes with various results the robot might need to narrow down its search to episodes where specific events have occurred. In this section we introduce several example reasoning queries on the two recorded experiments: A. Pancake making and B. Pizza making.

Below we present PROLOG queries for loading the recorded abstract events, in form of OWL files, and to connect to the low-level database. We then include general predicates used for reasoning on the two experiments (checking if specific events occurred, objects acted on, contacts between objects, etc.):

```
% load experiments into the knowledge base
?- load_experiments('path_to_owl_files').

% connect to the raw database
?- connect_to_db('db_name').


% get events which occured in the experiments
occurs(EpInst, EventInst, Start, End) :-
      rdf_has(EpInst, knowrob:'subAction', EventInst),
      rdf_has(EventInst, knowrob:'startTime', Start),
      rdf_has(EventInst, knowrob:'endTime', End).

% get a given event type
event_type(EventInst, EventType) :-
      rdf_has(EventInst, rdf:type, EventType).

% check object acted on
acted_on(EventInst, ObjActedOnType) :-
      rdf_has(EventInst, knowrob:'objectActedOn',
              ObjActOnInst),
      rdf_has(ObjActOnInst, rdf:type, ObjActedOnType).

% check particle type
particle_type(EventInst, ParticleType) :-
      rdf_has(EventInst, knowrob:'particleType', Part),
      rdf_has(Part, rdf:type, ParticleType).

% check objects in contact
in_contact(EventInst, O1Type, O2Type) :-
      rdf_has(EventInst, knowrob_sim:'inContact', O1),
      rdf_has(EventInst, knowrob_sim:'inContact', O2),
      O1 \== O2,
      rdf_has(O1, rdf:type, O1Type),
      rdf_has(O2, rdf:type, O2Type).

% check that Obj 1 is only in contact with Obj 2
only_in_contact(EventInst, O1Type, O2Type) :-
      findall(_, in_contact(EventInst, O1Type, _),
              AllCont),
      length(AllCont, Len), Len == 1,
      [Head|_] = AllCont,
      rdf_equal(Head, O2Type).
```

In the following we present some specific query examples for each experiment.

A. Pancake Making Scenario

In the following we present PROLOG rules checking if the pancake was successfully created and its quality (roundness). The rules start by verifying that the mondamin (batter) bottle has been grasped, afterwards a particle translation event has occurred (pouring the batter), and after the pouring all the particles landed on the oven surface (no spills). Afterwards the spatula should be grasped, followed by a sliding under the pancake event and a successful flip.

```
% grasp mondamin bottle
grasp_mondamin(EpInst, EventInst, Start, End) :-
      % get events which occurred in the experiments
      occurs(EpInst, EventInst, Start, End),
      % check for grasping events
      event_type(EventInst, knowrob:'GraspingSomething'),
      % check object acted on
      acted_on(EventInst, knowrob:'Mondamin').

% particle translation event
particle_transl(EpInst, EventInst, Start, End) :-
      % get events which occurred in the experiments
      occurs(EpInst, EventInst, Start, End),
      % check for particle translation
      event_type(EventInst,
        knowrob_sim:'ParticleTranslation').

% liquid is only in contact with the oven
liquid_only_on_oven(EpInst, EventInst, Start, End) :-
      % get events which occurred in the experiments
      occurs(EpInst, EventInst, Start, End),
      % check for touching situation
      event_type(EventInst,
        knowrob_sim:'TouchingSituation'),
      % check obj 1 is only in contact with obj 2
      only_in_contact(EventInst,
        knowrob:'LiquidTangibleThing',
        knowrob:'PancakeMaker').

% grasp spatula
grasp_spatula(EpInst, EventInst, Start, End) :-
      % get events which occurred in the experiments
      occurs(EpInst, EventInst, Start, End),
      % check for grasping events
      event_type(EventInst, knowrob:'GraspingSomething'),
      % check object acted on
      acted_on(EventInst, knowrob:'Spatula').

% slide under with the spatula
slide_under(EpInst, EventInst, Start, End) :-
      % get events which occurred in the experiments
      occurs(EpInst, EventInst, Start, End),
      % check for touching situation
      event_type(EventInst,
        knowrob_sim:'TouchingSituation'),
      % check objects in contact
      in_contact(EventInst, knowrob:'Spatula',
        knowrob:'PancakeMaker').

% check if the given model has been flipped
check_model_flip(EpInst, Model, Start, End, Flip) :-
      % initialize the "virtual knowledge base" iface
      mongo_sim_interface(MongoSim),
      get_raw_coll_name(EpInst, CollName),
      set_coll(CollName),
      # call to the Java function
      jpl_call(MongoSim, 'CheckModelFlip',
               [Start, End, Model], Flip).
```

The last predicate *check_model_flip* checks if the pancake has been flipped using the before mentioned "virtual knowledge base". This initializes a special query interface which allows the raw data to be included in the PROLOG query. The predicate calls a Java method, *CheckModelFlip*, using

the Java Prolog Interface[3] (JPL) which checks if at the two given timestamps the pancake particles have opposite normal vectors of their corresponding planes (flip occured).

Between the queries the events need to be verified that they happen at the right time and in the right order. For example the particle translation (pouring) event should happen while the bottle is being grasped. Otherwise it might be that the bottle has been knocked over and the batter has been spilled. Here are some examples of the queries:

```
..
%particle transl happened during the container grasp
comp_duringI(ParticleTranslEvInst, GraspMondaminEvInst),
..
%container grasping overlaps the liquid contacts
comp_overlapsI(GraspMondaminEvInst, LiquidContactEvInst),
..
%container grasp happened before the spatula grasp
comp_beforeI(GraspMondaminEvInst, GraspSpatulaEvInst),
..
```

Using the aforementioned predicates we will get access to the episode instances where the pancake was successfully created. The robot could then query for its specifically required information (trajectories, poses) from the raw database. In Figure 4 we have chosen to display an example of accessing data of such a successful episode. We visualize the spatula head trajectory (green-blue arrows - representing the normal vector of the spatula head) while sliding under the pancake and flipping it. We also queried for the pancake particles positions (red spheres) during flipping, and for visualization purposes we increased the visualization step to around 100*ms*.

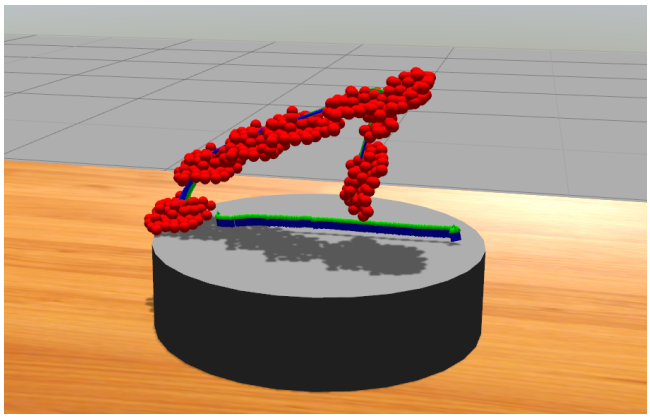


Fig. 4: Spatula and pancake trajectories during flipping

In order to find out the quality of the pancakes we added a predicate which checks if the poured batter ended up in a round shape:

```
% return the roundness of the pancake [0 - 1]
get_pancake_roundness(EpInst, Model, Ts, Roundness) :-
        mongo_sim_interface(MongoSim),
        get_raw_coll_name(EpInst, CollName),
        set_coll(CollName),
        jpl_call(MongoSim, 'GetPancakeRoundness',
                 [Ts, Model], Roundness).
```

For this, at the end of the pouring event we query the database for the position of each pancake particle. The shape is then evaluated using Principal Component Analysis (PCA) from Weka [8]. This is done by computing the covariance matrix of the particles' 3D points and performing the eigen decomposition of the matrix. This returns 3 eigen value-vector pairs. The smallest eigenvalue is going to be the thickness of the pancake and its corresponding vector is the normal to the pancake surface. The following two eigen values represent the variance of the pancake, in the other two dimensions, along their corresponding vectors. The ratio of these two values, the larger one being the denominator, gives a measure of the roundness of the data. A value close to 0 represent a highly elliptical form, and values close to 1 approach the shape of a circle.

In order to have more variance in our experiments, at one point we asked the users to try to make long shaped pancakes when pouring. In Figure 5 we have visualized various queried shapes of pancakes from our episodes. In the first line we can see the pancakes with roundness between $0-0.25$, in the middle row between $0.25-0.6$, we can notice that the shapes already start to be usable. In the final row we added pancakes with the shape close to 1. We also visualized the pouring trajectory of the bottleneck, green-blue arrows pointing towards the outside of the bottle. We can notice that for the long shaped pancakes the trajectory is varying in the XY plane (in the pictures the X axis is pointing forward, Y to the left, and Z being the height). We can also notice that height variance is not a very important factor for the roundness when pouring the batter. This can be seen in the bottom right picture, where we have a round pancake event though the bottle was moving in the Z axis.

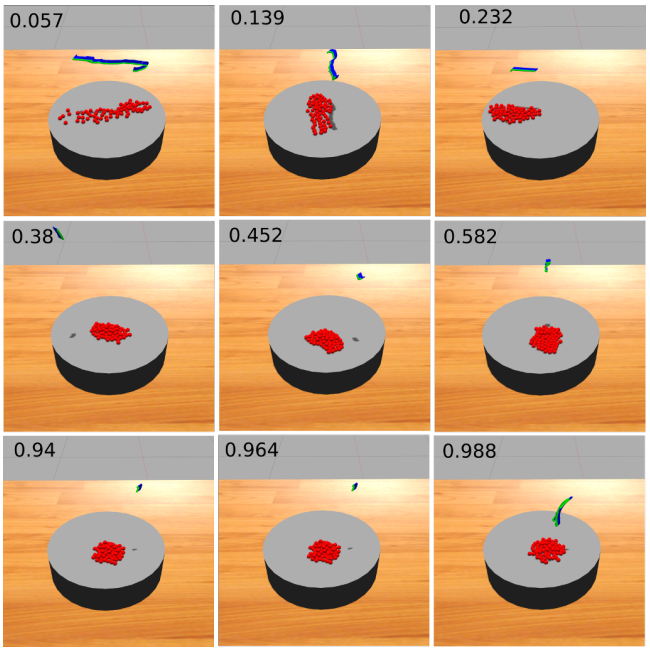


Fig. 5: Pancake roundness after pouring using PCA

In the following Figure 6 we decomposed the pouring trajectories for 30 episodes (in order not to clutter the visualization), querying for 10 roundness values between $[0-0.25]$ - Red, 10 between $[0.25-0.6]$ - Blue and 10

[3]http://www.swi-prolog.org/packages/jpl/

between $[0.6-1]$ - Green. We can notice that the red trajectories can drift away to up 10-15 cm from their starting points, while the others stay relatively constant.

This information can be very beneficial if the robot is specifically asked to pour a round pancake. It can reason from the episodes with round pancakes and observe that the bottle top should not be moving while tilting.

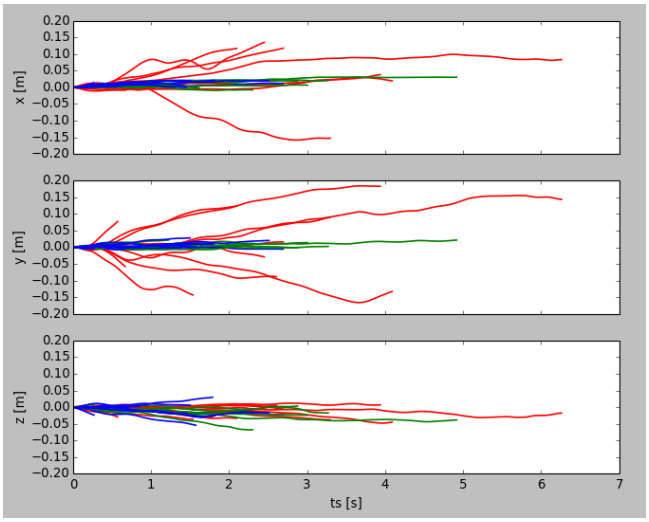


Fig. 6: Bottle pouring trajectory for pancakes with roundness: $[0-0.25]$ - Red, $[0.25-0.6]$ - Blue, $[0.6-1]$ - Green

B. Pizza Making Scenario

For the pizza scenario the user was presented with a flattened pizza dough, a container with pizza sauce, and two bowls of toppings: cheese and champignon. A correct pizza making scenario would include pouring the sauce on the dough, without spilling, spreading the sauce on the dough, again without spilling, and with the use of the spoon to scatter toppings over the pizza. In the following we will introduce the predicates used for recognizing a correct pizza making action. Most of the predicates are re-used from the previous scenario, for this reason we will only write their declarations.

```
% grasp the container with the sauce
grasp_cup(EpInst, EvInst, Start, End) :- ..

% particle translation from container
pour_sauce(EpInst, EvInst, Start, End) :- ..

% sauce is only in contact with the pizza
sauce_contact_overl(
        EpInst, EvInst, OverlInst, Start, End) :-
      % get events which occurred in the experiments
      occurs(EpInst, EvInst, Start, End),
      % check for touching situation
      event_type(EvInst, knowrob_sim:'TouchingSituation'),
      % container grasping overlaps the sauce contacts ev
      comp_overlapsI(OverlInst, EvInst),
      % check that Obj1 is only in contact with Obj2
      only_in_contact(EvInst, knowrob:'Sauce',
          knowrob:'Pizza').

% grasp tool
grasp_spoon(EpInst, EvInst, Start, End) :- ..

% topping particles leaving their bowl
topping_transl(EpInst, EvInst, DuringEvInst) :-
      % get events which occurred in the experiments
      occurs(EpInst, EvInst, _, _),
      % check for particle translation
      event_type(EvInst,
          knowrob_sim:'ParticleTranslation'),
      % transl happened while topping on the spoon
      comp_duringI(EvInst, DuringEvInst).

% check for the whole add topping event
add_topping(EpInst, EvInst, DuringEvInst, Topping) :-
      % get events which occurred in the experiments
      occurs(EpInst, EvInst, _, _),
      % check for touching situation
      event_type(EvInst, knowrob_sim:'TouchingSituation'),
      % topping is in contact with the tool
      in_contact(EvInst, Topping, knowrob:'Spoon'),
      % topping manipulation happened during tool grasp
      comp_duringI(EvInst, DuringEvInst),
      % during topping scatter ONE topping_transl occured
      findall(TranslInst, topping_transl(
          EpInst, TranslInst, EvInst), AllTopTranslInst),
      % check the size for strictly one
      length(AllTopTranslInst, Len), Len == 1.

% get a list of all the topping events
add_toppings_during(
    EpInst, DuringEvInst, Topping, AllTopEvInst) :-
      % check for all topping events of the given type
      findall(AddToppingEvInst,
              add_topping(EpInst,
                AddToppingEvInst, DuringEvInst, Topping),
              AllTopEvInst),
      % check that at least once topping has been added
      length(AllTopEvInst, AllToppingLength),
      AllToppingLength > 0.
```

The above predicates can recognize a successful pizza making task. We make sure no sauce particles are poured next to the pizza, or spread off the pizza. The add toppings events are more general since the user can choose the order on how to add them, might even need multiple runs to get the right amount out. Hence, the events are all stored in an 'add topping event' list. Further we check for spreading (smear) sauce event, this should happen before adding the first topping, and there might be multiple contact events between the spoon head and the pizza. So we take the smearing event as the start of the first contact, and the end of the last one, all before adding any toppings:

```
% pizza contact when smearing sauce
smear_pizza_contact(EpInst, EvInst, BeforeEvInst) :-
      % get events which occurred in the experiments
      occurs(EpInst, EvInst, _, _),
      % check for touching situation
      event_type(EvInst, knowrob_sim:'TouchingSituation'),
      % check objects in contact
      in_contact(EvInst, knowrob:'Spoon', knowrob:'Pizza'),
      % contact occurred before the add toppings
      comp_beforeI(EvInst, BeforeEvInst).

% smear sauce happens before add toppings
smear_sauce_before(EpInst, BeforeEvInst, Start, End) :-
      % allspoon-pizza contacts before add topping
      findall(PizzaContactInst,
              smear_pizza_contact(
                EpInst, PizzaContactInst, BeforeEvInst),
              AllSmearPizzaContacts),
      % get the first and last event from the list
      first_last_from_list(
        AllSmearPizzaContacts, FirstEv, LastEv),
      % get the start from the first event
      occurs(EpInst, FirstEv, Start, _),
      % get the end from the last event
      occurs(EpInst, LastEv, _, End).
```

In Figure 7 we have visualized the spreading of the pizza sauce from a successful episode. Blue spheres represent the sauce particles poured on the dough and red the resulting particles positions after spreading them with the spoon. The arrow trajectory represents the spoon head movements while spreading the sauce.

V. RELATED WORK

In [9] the authors present a memory system for cognitive robots. While executing tasks with known intentions, the

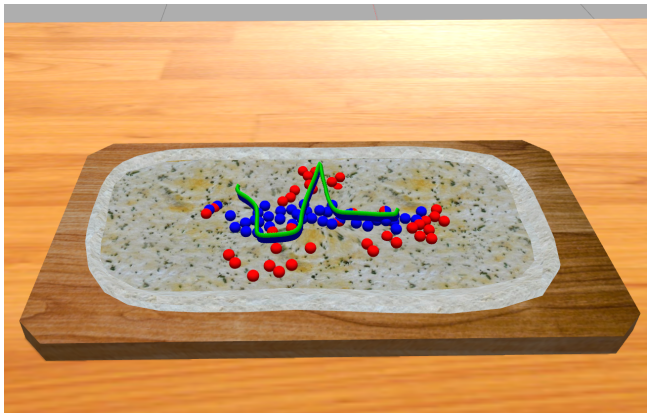


Fig. 7: Spreading the sauce on the pizza, blue particles represent the initial poured position and red after spreading

system stores information about probable failures and frequent pitfalls with the scope of making improvements when executing identical or similar tasks in the future. They are storing in a knowledge base two streams of events, one being high level symbolic hierarchical task structure of executed plans. And the other low level sensor data stored indirectly in the knowledge base. The system is similar to the one presented in this article, but instead of storing memories from past execution of the robot, we get human task knowledge by letting them execute tasks in a virtual environment.

In [10] a reasoning system about commonsense knowledge and naive physics is presented in the context of everyday activities. The authors are using to PROLOG to assert initial conditions to given scenarios and using a physics-based simulator they get temporal projections about parameterized robot control programs. Various experiments were conducted were formal parameters of robot control programs were selected from various ranges. Their results were further evaluated with respect to their outcome. The presented framework uses a simulator as well to receive various knowledge on the effects of actions, however in order to receive the required parameters for the control program they need to run multiple simulations which can be a slow process. In comparison we have the knowledge base already build, and if the information is available, then it is quickly accessed.

The interactive cooking simulator [11] is aimed for understanding the various stages of cooking (e.g. temperature changes, protein denaturation, burning process). The simulator can visually express the various states of cooking. Their proposed simulation speed is fast enough to enable interactive cooking. In our simulation the effects of cooking are ignored, since we aimed more in the movements required to complete a task. However, information about the cooking stages can be of great help for learning various other parameters from humans.

VI. CONCLUSION AND FUTURE WORK

In this paper we presented a reasoning system for cognitive agents acting in the context of every day scenarios with the purpose of mining commonsense and naive physics knowledge. We have extended a knowledge representation and reasoning system with a new knowledge source, generated by humans executing various tasks in virtual scenarios. We have used two different cooking scenarios as experiments for collecting data and testing examples. We have introduced various PROLOG queries as examples to reason on the introduced knowledge type. All the experiments and the presented reasoning queries are available on the OPENEASE [12] website[4].

For future work we plan to overcome the numerous tweaks required to simulate liquids/soft objects when using only a rigid body physics engine. We will looking into various game engines that supports particle based simulation out of the box. This can greatly improve the quality of a simulation, yielding data more close to the reality, and reduce the time required to set up new scenarios. The event detection can be extended to support more specific events such as: supported by, inside of, etc. By testing the framework on different scenarios we can look into how to generalize the predicates even further. Having Weka integrated we can also test various machine learning algorithms on the collected data, which the robot can use during execution.

REFERENCES

[1] 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.

[2] L. von Ahn and L. Dabbish, "Designing games with a purpose," *Commun. ACM*, vol. 51, no. 8, pp. 58–67, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1378704.1378719

[3] R. S. Johansson and J. R. Flanagan, "Coding and use of tactile signals from the fingertips in object manipulation tasks," *Nature Reviews Neuroscience*, vol. 10, no. 5, pp. 345–359, 2009.

[4] 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)*, pp. 2149–2154, 2004.

[5] 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.

[6] W3C, *OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax*. World Wide Web Consortium, 2009, http://www.w3.org/TR/2009/REC-owl2-syntax-20091027.

[7] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, Nov. 1983. [Online]. Available: http://doi.acm.org/10.1145/182.358434

[8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, pp. 10–18, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278

[9] J. Winkler, M. Tenorth, A. K. Bozcuoglu, and M. Beetz, "CRAMm – memories for robots performing everyday manipulation activities," *Advances in Cognitive Systems*, vol. 3, pp. 47–66, 2014.

[10] L. Kunze and M. Beetz, "Envisioning the qualitative effects of robot manipulation actions using simulation-based projections," *Artificial Intelligence*, pp. –, 2015.

[11] F. Kato and S. Hasegawa, "Interactive cooking simulator: Showing food ingredients appearance changes in frying pan cooking -," in *Proceedings of the 5th International Workshop on Multimedia for Cooking and Eating Activities*, ser. CEA '13, 2013.

[12] M. Beetz, M. Tenorth, and J. Winkler, "Open-EASE – a knowledge processing service for robots and robotics/ai researchers," in *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, Washington, USA, 2015.

[4]http://www.open-ease.org/