Learning Motion Parameterizations for Fetch and Place Tasks from Observing Humans in Virtual Environments

Gayane Kazhoyan, Alina Hawkin, Sebastian Koralewski, Andrei Haidu and Michael Beetz*
{kazhoyan, hawkin, seba, haidu, beetz}@cs.uni-bremen.de

***Abstract*— In this paper, we present an approach and an implemented pipeline for transferring symbolic and subsymbolic data acquired from observing humans performing fetch and place tasks in virtual environments onto the robot, and adapting the data accordingly to achieve successful task execution in the real world. We demonstrate our pipeline by inferring seven different motion parameters in the context of setting a simple breakfast table. We propose an approach to learn general motion parameter models and discuss, which parameters can be learned at which abstraction level.**

I. Introduction

Robots acting in real-world environments, such as human households, require a vast amount of various knowledge to be able to execute their tasks, including naive physics, commonsense and task-specific knowledge. For example, consider the task of setting a table for breakfast. The robot needs to know which objects are involved in the task, where to find them, how to stand to have an object in the field of view, how to grasp objects, where to stand to be able to reach them, what is the appropriate table setting configuration in the particular context, etc.

This knowledge is difficult to obtain: one either has to write specialized reasoners for each task, or learn from own experience with a trial and error approach over all the possible solutions in the domain of the problem. Alternatively, the robot could learn from humans through imitation.

Virtual reality (VR) technology nowadays is getting very popular and easily accessible. VR systems allow humans to interact with a virtual environment in a natural and intuitive way. Logged data of humans executing tasks in VR is a powerful source of everyday activity knowledge that can be used to teach robots. As opposed to more traditional camera-based human motion tracking in real world, using VR gives the advantage of easily varying the environment and task scenarios, it provides highly accurate ground truth data, and, finally and perhaps most importantly, it has the advantage of having direct access to the underlying world physics. This means that contacts, supporting relations, visibility, occlusions, furniture states (drawer is open X cm), all force-contact events (e.g., hand touched cup, cup lost contact with supporting surface) can be directly read out from the physics engine, which is otherwise difficult to estimate in the real world without using accurate internal and external sensors.

Using a mechanism for automatic knowledge extraction from VR data, the robot can learn to answer questions necessary to parameterize its motions. In this paper, we present an approach and an implemented pipeline for transferring symbolic and subsymbolic data acquired from VR human data onto the robot and for adapting it accordingly to achieve successful task execution (see Figure 1). This requires to solve a correspondence problem between the observed actions and robot actions, and how to transform the observed data into executable one. The data cannot be transferred as is, due to the differences between the virtual and real environments and objects in them, the physical body and capabilities of robots and humans, and the contexts of tasks performed in VR and the real world. We perform an experimental analysis of which data is possible to transfer and present approaches to generalize the data such that it becomes applicable to robot's own environment and task context. Additionally, we discuss the limitations of data transfer and what is impossible to learn from VR demonstrations with the state of the art techniques.

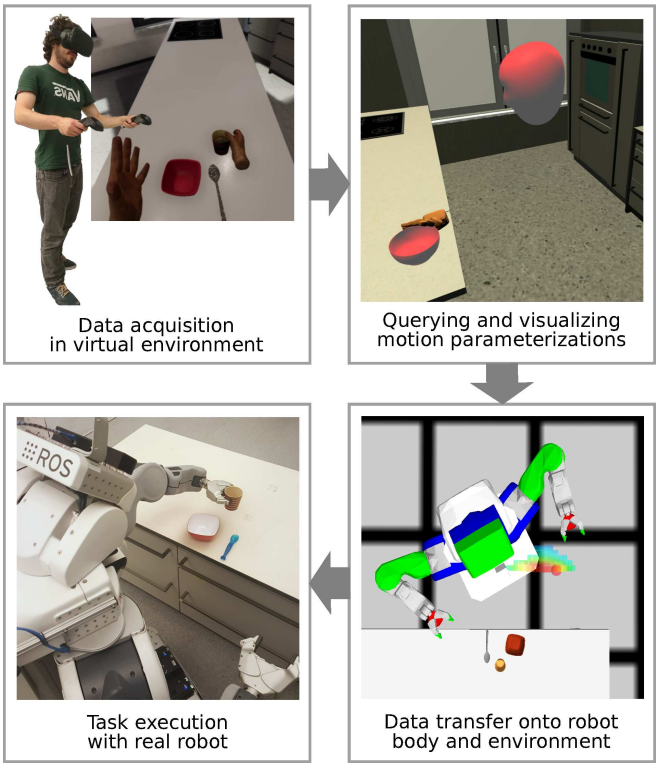

Fig. 1. Process pipeline of the presented approach: humans perform experiments in VR, the observations are logged into a subsymbolic database and symbolic ontology, motion parameter data is transferred onto the robot and its environment in plan projection simulator, and, finally, the learned parameters are used on the real robot for its tasks.

*The authors are with the Institute for Artificial Intelligence, University of Bremen, Germany.

An important property of our approach is that we transfer the knowledge from humans to robots in a white box manner. As opposed to end-to-end approaches, where the learned model is very high dimensional and is a black box, we factorize our problem into smaller subproblems, such that the learned models are directly associated with the corresponding motion parameters, and the causal effects of good or bad parameter choices are easy to track.

The contributions of this paper to the state of the art are:

- a pipeline for transferring motion parameterizations from VR onto the robot, which generalizes over different environment setups;
- an analysis of transferability of data acquired in VR onto the robot in the context of fetch and place tasks.

We demonstrate our pipeline by inferring answers to seven questions, including semantic information such as that bowls should be grasped from the top side, and subsymbolic data such as geometric arrangements of objects on a table for setting it for a meal or the distribution of locations, where the robot should stand to be able to reach an object. We prove that the transferred data is of high quality and enables the robot to execute a mobile manipulation task of setting a breakfast table in a real world kitchen environment.

II. Related Work

Teaching robots to perform tasks based on imitation learning from observing humans and using imitation to bootstrap reinforcement learning problems is a wide-spread approach in robotics [1], [2]. In this section, we review related work in mobile manipulation and learning from virtual environments.

Welschehold et al. [3], [4] learn manipulation action trajectories of tasks like opening and closing drawers by observing humans acting in the real world and adapting the data to the robot's capabilities through a hyper-graph optimization algorithm. The data generalizes towards a new body that executes the task but it does not generalize to different environments, as the trajectories are learned for a specific furniture piece. Additionally, Welschehold et al. learn very specific motion parameters, whereas, in this paper, we present a general pipeline that can learn any motion parameterization based on an extensive log of everything that the human does in the virtual environment, and we learn a full set of parameters, as many motion parameterizations of a certain task are interdependent (see Section IV).

Zhang et al. [5] use a VR setup for teleoperating a PR2 robot to perform a manipulation task, such as attaching a wheel to a toy plane, in order to use the gained data for deep imitation learning. Teaching by teleoperation can be costly, as it requires access to robot hardware. On the other hand, collecting data from humans interacting with the environment in VR is a promising approach to crowdsource data collection for robots. In our paper, VR is used to create an environment for a human to naturally execute tasks. However, this creates the problem of mapping between the human and robot bodies, which we consider in this paper.

An example of learning directly from human experiments in VR is [6] by Bates et al. Their setup and the fundamental idea is similar to ours, however, they do not learn low-level motion parameterizations from human demonstrations but apply automatic motion segmentation algorithms to infer the sequence of actions the human has performed, in order to generate a high-level plan for the robot. In this paper, we go all the way to the low-level motion parameters up until the limitations of what is possible to learn realistically from a simulated environment.

Dyrstad et al. [7] utilize VR to teach a robot to grasp real fish by observing humans perform fish grasping actions in a virtual environment. The particularity of their approach is that the humans control a robot gripper in VR, such that the complication of mapping between the two bodies does not arise. They use the data acquired in VR to generate a large amount of synthetic data, which is used to train a CNN.

In our paper, we do not concentrate as much on the learning algorithms themselves as on the general pipeline that can learn answers to various questions, supporting both symbolic and subsymbolic learning models. We demonstrate a number of simple algorithms that can be used to learn generalized models, which we found work best with our data, but the proposed pipeline can be combined with any other state of the art learning approach as well.

III. Data Acquisition by Observing Humans in VR

In this section, we present the adapted knowledge acquisition pipeline from our previous work [8] that is used to collect robot understandable data from observing humans in virtual environments. We explain how the data is generated, stored, and accessed by the robot.

Fig. 2. Collecting symbolic and subsymbolic data in VR

Figure 2 visualizes the process of collecting the data: we ask the human user to execute an underdetermined task in the virtual environment, and during execution we automatically log subsymbolic (trajectories, poses) and symbolic (actions, events) data. The virtual environment is connected to the knowledge base of the robot, by maintaining a mapping of every entity from the virtual world to its corresponding class in the robot's ontology [9]. Thus, every time an action (such as grasping) or a physics event (such as contacts between objects) is logged, we automatically know the types of the objects that were participating in the logged event.

To interact with the virtual world we use an off-the-shelf VR headset with hand tracking controllers. The tracked poses of the headset and of the controllers are mapped onto the virtual head and the hands of the user. Since the

interaction with the environment is force-based (e.g., one needs to pull/push against handles to open/close drawers), the mapping of the hands' movements to the controllers' tracked position is done using force-based PID controllers.

During task execution the data is saved in two places: (1) the low-level (subsymbolic) data is stored with high-frequency in a MongoDB database, representing the world state at every timestamp, which allows us to re-create the complete state of the world at any given timestamp; (2) the high-level (symbolic) data, such as contact events or grasping actions, are stored in an OWL format, making it available to reason upon in the robot's knowledge base.

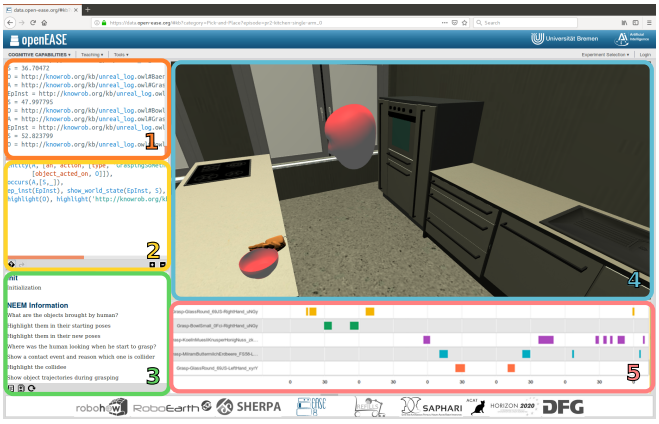

Fig. 3. openEASE web interface with visualization of data acquired in VR, where: (1) shows the terminal results of executed queries; (2) is the input panel for queries; (3) shows predefined queries stated in natural language; (4) is a 3D rendering of the results of the query; and (5) is an interactive timeline visualization of the logged events.

To visualize the data one can use openEASE[1] [10], which is a web interface capable of accessing the semantically logged data, with the possibility to graphically visualize it. Figure 3 shows the visualization of an example query from a table setting scenario.

IV. Parameterizations of Fetch and Place Tasks

In this section, we explain the fetch and place plans that control robot execution in the real-world environment, based on the knowledge extracted from the VR data. In our system, the plans are written with the so-called *action descriptions* [11], which are abstract underspecified descriptions of the actions the robot needs to execute. For example, the plan can contain the following command to perform an action:

```
(perform
  (an action (type fetching)
             (object (the object
                          (type cup)
                          (pose some-pose)))))
```

which describes the action of fetching a cup object that has already been found in the environment. When execution reaches a command to perform an underspecified action description, reasoning queries are asked to the knowledge system to infer the missing parameters of the action, resulting in a fully specified and executable action description. In our example, the missing parameters would be the arm, with which to grasp, if the robot has multiple arms, the base location where to stand, such that the robot can successfully reach the object with the given arm, as well as the grasp pose for the specific object, including the grasp point and the orientation of the gripper, such that the robot can reach the object from the given base position. The question of how to grasp an object requires the robot to have knowledge of kinematics of its body, the properties of the object (e.g., heavy, fragile) and the context (e.g., do not touch the inside of a cup if it is going to be used for drinking). Such knowledge is difficult to program. Our approach in this paper is to make use of human's commonsense and task knowledge instead of programming it directly into the robot.

Our fetch and place plans are designed as hierarchical structures, which are comprised of a set of calls (sequential or concurrent) to perform action descriptions. Performing an action description is a two-step process: first, the missing motion parameterizations are inferred, then, the plan associated with the given action description is executed with the inferred parameters. The plan, in its turn, may contain calls to perform lower-level action descriptions. This results in a hierarchy of plans, where the leaves are calls to perform lowest-level robot actions that are executed by directly calling robot's hardware controllers or the perception system.

The highest-level action in our fetch and place plans is the *transporting* action. The plan that executes it contains calls to perform three child actions: *searching*, *fetching* and *delivering*. Table I lists their motion parameters.

Plan	Direct parameters	Parameter type
search	object-likely-location	subsymbolic
	base-location-for-perceiving	subsymbolic
fetch	base-location-for-grasping	subsymbolic
	grasp	symbolic
	arm	symbolic
deliver	object-placement-location	subsymbolic
	base-location-for-placing	subsymbolic

TABLE I
Parameters of the child actions of the transporting plan.

The *searching* action looks in the environment for an object that satisfies the constraints given in the abstract object description. For example, performing *(an action (type searching) (object (an object (type cup))))* results in the robot searching for a cup object in its environment. The main parameters of *search* are: (1) *object-likely-location*, which is the likely location of where the object could be, including semantic information on what kind of surface or in what kind of container it can be, and the exact point on the surface or container to look at; and (2) *base-location-for-perceiving*, which is the pose at which to position robot's base such that the robot can perceive *object-likely-location* from a near-enough distance and without occlusions. As one can see, *base-location-for-perceiving* depends on the choice of *object-likely-location*.

The *fetching* action assumes that the object to fetch has already been found and the exact pose is known. *base-location-for-grasping* is the pose for the robot to stand, such

[1] http://open-ease.org

that it is able to reach the object and the trajectory does not result in a collision. *grasp* is one of the symbolic values *top*, *left-side*, *front*, *handle*, etc., which corresponds to the side of the object that the robot is going to grasp from. The actual subsymbolic grasp poses are defined in our system in the knowledge base, such that at runtime the robot only needs to choose a suitable grasp from the list of available ones for the given object. The *arm* parameter of the *fetching* action defines if the robot grasps with its *left* or *right* arm.

For the *delivering* action the following two parameters have to be inferred. The first one, *object-placement-location*, defines on which symbolic type of surface or container the object has to be placed and the exact pose on that specific surface/container, e.g., the location for a spoon would be on a *DiningTable* surface, specifically at a pose right of the bowl and aligned with the table. Depending on *object-placement-location*, *base-location-for-placing* is the robot base pose such that the object placement pose is reachable.

An important property of our plans is that they contain carefully designed failure handling strategies. This implies that the inferred motion parameters of the plans do not have to always be correct. Instead, when a parameter results in a failure, the failure handling strategies pick another parameter from the list of suggestions from the (VR-based) knowledge base, and retry the plan with this new solution. As a result, the better the quality of the VR data, the less retries and, therefore, runtime the robot requires to successfully execute the task. Below is a code snippet showing a simple failure recovery strategy for choosing a different grasp when picking up an object fails:

```
1  (with-failure-handling
2      (perform (an action (type picking-up)
3                          (object ?object)
4                          (grasp ?grasp)))
5    (manipulation-failure (failure-instance)
6      (setf ?grasp (next ?grasp))
7      (if ?grasp (retry))))
```

We wrap the command for performing the action (lines 2-4) in a failure handling guard (line 1), which, when a failure of type *manipulation-failure* happens (line 5), sets the value of variable *?grasp* to the next value from the grasp generator (line 6), and if there is a next value, retries to perform the action (line 7). *with-failure-handling* and *retry* are language constructs from our domain-specific robot programming language CPL [12].

V. Transferring Human Data onto the Robot and Its Environment

We access our knowledge databases through a Prolog interface. Let us consider, how the seven parameters from Table I are inferred from the data acquired in VR.

To infer *object-likely-location* when searching for an object, we use the relative location of the object to its supporting surface that has been observed in the human data. The object the robot is searching for is described by its ontological type, e.g., in *(an action (type searching) (object (an object (type bowl))))*, *bowl* is any object of the corresponding class in the ontology. Thus, our first constraint is that the specific object instance from the VR data is of type *bowl* or any of its subclasses:

```
subclass_of(Object, 'bowl'),
```

Next, we find all the events from all the experiment episodes, where the human grasped an object of the given type, and take the timestamp of the beginning of that event:

```
episode(Episode),
occurs(Episode, Event, StartTime, EndTime),
type(Event, 'GraspingSomething'),
property(Event, 'objectActedOn', Object),
```

Taking the time of the beginning of the event (*StartTime*) ensures that the object is at its initial location where the human has found it, and not, e.g., at the location where it has been placed later on.

To find the supporting surface, we look up a *TouchingSituation* event, which involves our object:

```
occurs(Episode, TouchEvent, TouchStart, TouchEnd),
type(TouchEvent, 'TouchingSituation'),
property(TouchEvent, 'inContact', Object),
property(TouchEvent, 'inContact', Surface),
not(Object==Surface),
```

The grasping event has to have happened while the contact situation has been active, otherwise we might get the contact event with the surface at the target location:

```
time_between(Start, TouchStart, TouchEnd),
```

Note, how we are using the force-contact events stored in the database to segment the continuous data stream of observations into discrete human action sequences. By doing so, we establish matches between human actions and their corresponding counterparts in the robot plans, to extract the relevant data for estimating a given motion parameter.

In order to make sure that our grasping event corresponds to the object being transported to the correct destination, we need to take into account the task context. In our scenario, the context is of setting a table, so all other actions, e.g., of cleaning it up afterwards, need to be disregarded. We filter the grasping events by looking at the destination surface of the event: if the destination is of type *DiningTable*, we assume that the action was to set a table:

```
type(ContactSurfaceAtEndOfGrasp, EndSurfaceType),
not(subclass_of(EndSurfaceType, 'DiningTable')
```

Now that we have constrained the grasping event to the context of our scenario and inferred the supporting surface of the object at the beginning of the object transport, we infer the location of the object relative to the surface. For that we need the object location and the surface location in the global coordinate frame:

```
actor_pose(Episode, Object, StartTime, ObjectPose),
actor_pose(Episode, Surface, StartTime, SurfPose).
```

The other parameters are inferred similarly. *base-location-for-perceiving* is inferred from the pose of the human relative to the object to perceive at the time point when the grasping event started. As we can only track the head of the human in VR and not his feet, we map from the human head pose to the robot's base location by, first, projecting the pose onto the floor, and then straightening the orientation to get rid

of the tilt of the head, such that the robot base is parallel to the floor. Additionally, as our robot's base is wider than the human feet by approximately 20 cm, we add a constant offset of 20 cm in the *-x* direction of the pose, i.e. towards the back. This is the only explicit offset that we use to adjust the human data to the robot body. When transferring the data onto a different robot body, this and only this value has to be adjusted towards the new robot platform.

base-location-for-grasping and *base-location-for-placing* are inferred similarly. *grasp* is inferred from the relative position of human's wrist with respect to the object, e.g., if the wrist was located above the object, it is a *top* grasp. *arm* is bound to the value *left* or *right* based on the hand, with which the human grasped the object.

object-placement-location is a parameter, which greatly depends on the task context. For example, the location of a spoon on a surface near the sink in the context of cleaning the table does not imply strict constraints on its orientation or position. On the other hand, in the context of table setting, the orientation of the spoon has to be aligned with the supporting surface. Additionally, if there is another object, such as a plate or a bowl, on the table, the position of the spoon is constrained to being right of the other object. This depends on the viewpoint of the person using the utensils, therefore, the location of the human also influences *object-placement-location* in this context. The relative positioning of the object on the table is, thus, calculated based on the bounding box of the supporting surface, the relative poses of the objects involved, as well as the human pose when placing the objects.

VI. Learning Generalized Models for Fetch and Place Tasks

In this section, we outline how the data generated by executing plans on the robot with VR-based inference can be used to learn general models of motion parameters. We show how three of the seven parameters from Table I can be inferred from learned models. The other four parameters are calculated directly from the VR data as a one to one mapping, without using a generalized model in between.

In [13] we inferred fetch and place action parameterizations from robot's own experience data. To generate the data, the robot utilized heuristics for plan parameterization. Heuristics allow to minimize the search space, however, they have to be handcrafted for each parameter inference task and are very prone to failures. In this paper, the heuristics have been replaced by VR-based inference. We reused the statistical models from our previous work and retrained them based on data acquired by the robot performing its plans, where motion parameterizations are inferred from VR.

In the context of a fetching action, to infer grasping poses and robot base locations that are expected to lead to successful action execution, we utilize a probability distribution, defined as follows:

$$\frac{P(S \mid \mathbf{GP}, \mathbf{RFF}, RP, OT, SF, ARM)}{\sum_{rp}^{\mathbf{RP}} P(S \mid \mathbf{GP}, \mathbf{RFF}, rp, OT, SF, ARM)}$$

where

$$\mathbf{GP} = \underset{GP}{\operatorname{argmax}}\, P(GP \mid \mathbf{RFF}, OT, SF)$$
$$\mathbf{RFF} = \underset{RFF}{\operatorname{argmax}}\, P(RFF \mid RP, OT, SF)$$

We determine the maximum probability of success S, given the robot base pose RP relative to the object, grasping pose GP represented as a discrete variable, object type OT, arm ARM and the object orientation represented as two discrete random variables – supporting face (SF) and robot facing face (RFF) – which take the same values as GP. We assume that the object orientation is always constrained by its supporting surface, thus, we represent it using the face of the object that is in contact with the supporting surface (SF) and the angle around the axis perpendicular to the surface. We discretize the continuous space of angles into four symbolic object faces, and the face, the normal of which points towards the robot, we call the robot facing face RFF. To infer the robot facing face (RFF), we are applying linear algebra, which allows us to determine it with a probability of 1.0.

We use Fuzzy Markov logic networks (FUZZY-MLN) [14] as a statistical model, to infer the discrete grasping pose GP for picking up an object. A general Markov logic network (MLN) is a model, which combines probabilistic graphical models and first order logic [15]. FUZZY-MLN enhances a general MLN by utilizing the fuzzy logic calculus during inference. The advantages of MLNs are that they can represent complex relations and that they are white box models, which allows to easily interpret the results of learning and understand the causal relationships between the learned features. Based on the discrete object pose representation, we are able to infer from the learned FUZZY-MLN the required grasping pose GP, which with highest probability results in the robot grasping the object successfully.

Having RFF and GP inferred, we are able to create a distribution, which represents the success probability to grasp the object based on the robot position. To calculate the success probability, we created a binary classifier, which labels if the fetching action will be successful or not given the evidence such as grasping pose, object type and robot base pose. Since, at this moment, we did not have enough data points from the virtual reality for deep learning approaches, we decided to use generative models (Gaussian naive Bayes) to represent the success probability. We have one Bayes classifier for each combination of ($OT \times GP \times RFF \times SF \times ARM$) to keep the models simple and to be able to perform fast reasoning.

The probability distribution for the fetching action has the advantage that it can be reused for delivering an object to a given location. We interpret the delivering action as a reverse of fetching, and, thus, represent the probability of success of a delivering action similarly to the fetching probability:

$$\frac{P(S \mid GP, RFF, RP, OT, SF, ARM)}{\sum_{rp}^{\mathbf{RP}} P(S \mid RFF, GP, rp, OT, SF, ARM)}$$

In the delivering action, the agent already has the object in the hand, so we consider the grasping pose as given.

Figure 4 shows the learned models for positioning the robot's base for fetching a bowl object and placing a cup.

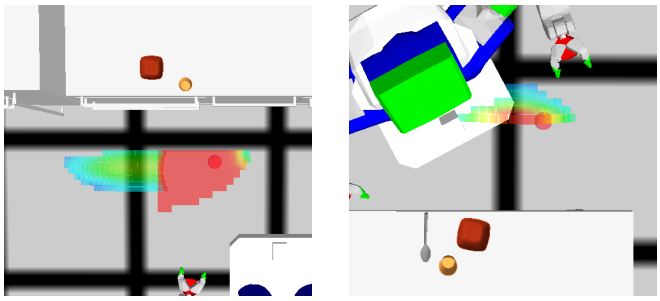

Fig. 4. Distribution for a robot base pose to (left) grasp a bowl with the left arm and (right) place a cup with the right arm, visualized as heat maps.

VII. Experimental Analysis

In this section, we empirically investigate the effectiveness of using data, acquired by observing humans performing tasks in VR, for executing mobile fetch and place tasks on a real robot. Specifically, we investigate the following:

1) Is it possible to successfully execute a simple table setting task on a robot by inferring motion parameters based on VR data?
2) How does VR-based motion parameter inference compare to hand-crafted heuristics?
3) How does the quality and quantity of the data collected by observing humans in VR affect robot performance?
4) How does the system scale to changes in robot's environment?
5) Can the system be applied to different robots?
6) Are human preferences reflected in the robot behavior?

The task in all the experiments is to set a simple breakfast table with three objects – bowl, spoon and cup – by bringing them from the sink counter to the dining table.

A. Data Acquisition in VR

We have recorded data in VR in separate batches, which differ in a number of properties, listed in Table II.

VR Data Batch ID	# Table Setting Episodes	Virtual Environment	Human Handedness
LikeReal	60	1 kitchen, similar to real	Right
3Var	20 * 3	3 kitchen variations	Right
3VarHalf	10 * 3	3 kitchen variations	Right
LeftSide	3 * 3	3 kitchen variations	Left

TABLE II

Batches of data acquired by observing humans in VR.

The first batch was recorded in a virtual environment that closely resembles robot's kitchen in the real world. The positions and orientations of the objects on the source and destination surfaces have been varied in each episode.

For the other batches, we have created three different kitchen environments, where the scale, position and orientation of the furniture pieces have been greatly varied, making sure to create environments as different from the real one as possible (see Figure 5). The third batch is simply a subset of the second batch, for evaluating quantitative effects of data on robot performance. In the fourth batch, the human user put the spoon only on the left side of the bowl, to investigate if human preferences are reflected in the robot behavior.

Fig. 5. The real environment and the three virtual ones. The position, orientation and scale of furniture in all four environments is different, such that no virtual kitchen is similar to the real one.

B. Experiments

To answer the questions, listed at the beginning of this section, we have executed the table setting task in many different variations, listed in Table III.

Experiment ID	# Table Setting Executions	Robot	Robot Environment	Inference Engine	VR Data Batch ID
Pr2HrSim	100	PR2	Simulated, similar to real	Heur.	–
Pr2VrSimLikeReal	100	PR2	Simulated, similar to real	VR	LikeReal
Pr2VrSim	100	PR2	Simulated, similar to real	VR	3Var
Pr2VrSimHalf	100	PR2	Simulated, similar to real	VR	3VarHalf
Pr2HrReal	20	PR2	Real	Heur.	–
Pr2VrRealLikeReal	20	PR2	Real	VR	LikeReal
Pr2VrReal	20	PR2	Real	VR	3Var
Pr2HrSimMod	100	PR2	Simulated, modified	Heur.	–
Pr2VrSimMod	100	PR2	Simulated, modified	VR	3Var
Pr2VrSimLeft	10	PR2	Simulated, similar to real	VR	LeftSide
Pr2VrSimModLeft	10	PR2	Simulated, modified	VR	LeftSide
Pr2VrRealLeft	3	PR2	Real	VR	LeftSide
BoxyHrSim	100	Boxy	Simulated, similar to real	Heur.	–
BoxyVrSim	100	Boxy	Simulated, similar to real	VR	3Var

TABLE III

Execution variations of table setting tasks.

Experiments were executed on a real-world robot, as well as in a fast simulation environment [16], randomly varying the position and orientation of the objects' initial location on the sink counter. The two robots that were used are PR2 and Boxy, which are mobile manipulation platforms with two arms. Boxy is bulkier than PR2 and has less

Experiment ID	Success Rate (%)	Success Rate Per Object (%)			# Non-recoverable Failures			# Recoverable Failures			Execution Time (s)
		Bowl	Cup	Spoon	Search	Fetch	Deliver	Percept.	Nav.	Manip.	
Pr2HrSim	33	81.5	44	90	0	0.3	0.545	11.89	4.5	44.26	16.3 ?
Pr2VrSimLikeReal	60	100	60	100	0	0.1	0.3	2.8	160.8	30.8	14.74
Pr2VrSim	75	95	85	95	0	0.1	0.15	6.5	304.9	24.1	17.9
Pr2VrSimHalf	33	83	77	52	0.02	0.7	0.15	9.79	1000.46	46.56	15.52
Pr2HrReal	50	60	80	90	0	0.5	0.2	11	14.9	61.7	373.62
Pr2VrRealLikeReal	90	100	90	100	0	0	0.1	4.2	203.1	53.5	420.3
Pr2VrReal	90	100	90	100	0	0.1	0	1.56	259.4	34.8	349.43
Pr2HrSimMod	12	38	63	68	0.1	1.03	0.18	14.57	45.47	63.04	13.25
Pr2VrSimMod	11	39	72	66	1.2	0.22	0	2.09	3627.39	4.74	21.77
BoxyHrSim	72.5	94	83.5	94	0	0.28	0.005	10.05	13.64	42.53	15.01
BoxyVrSim	48	84	66	84	0	0.62	0.04	7.12	836.3	49.54	24.83

TABLE IV

SUCCESS RATES AND EXECUTION TIME OF EXPERIMENTS FROM TABLE III, AVERAGED OVER THE NUMBER OF EXECUTIONS. EXECUTION TIME IS THE AVERAGE DURATION OF ONE SUCCESSFUL OBJECT FETCH AND PLACE.

reachability (see Figure 6(left)). In order to show that the system scales towards changes in robot's environment, we designed a modified kitchen for the simulator, which is unlike the real kitchen or any of the VR ones (see Figure 6(right)).

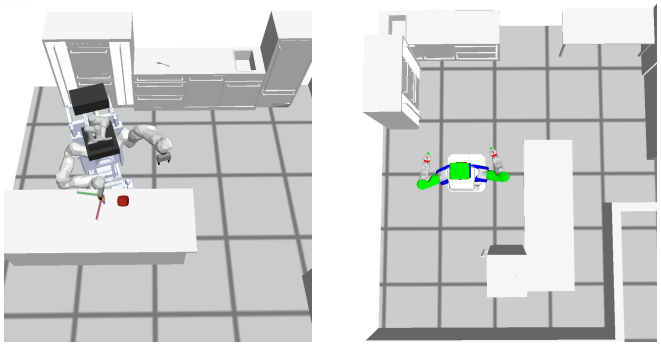

Fig. 6. A different robot (left) and a different robot environment (right).

As a baseline for comparison, we have used handcrafted heuristics to infer missing motion parameterizations [16]: to calculate object likely locations and destination poses, we use bounding boxes of supporting surfaces; for visibility calculations, offscreen rendering is used; for reachability, a simple robot-specific circular region is generated; and the choice of the arm and the grasp is done by random sampling.

A few experiments have been performed with the *LeftSide* VR data batch loaded, in order to show that the robot is able to not only learn suitable table setting arrangements for the given task context but also takes into account the preferences of the human, who collected the data in VR (see Figure 7.

Table IV shows the results of the experiments, averaged over the number of table setting executions per experiment[2]. By looking at action execution success rates, we can conclude that our system successfully learned the motion parameters of a transporting action (see Table I), including (1) where to look for objects in the environment, (2) where to position the robot in the environment such that the object is visible and not occluded, (3) where to position and how to orient the robot in order to reach an object, (4) from which side to approach the object with the gripper, (5) which arm to use for picking up the object, (6) where to stand in the environment to place the object, and, finally, (7) how to arrange objects in the given context, in our case, table setting, also based on the preferences of the human user.

[2]The specs of the simulation PC are: Ryzen7 8 core 3.2GHz processor, 32 GB RAM and an NVidia GeForce GTX 1050 Ti graphics card. The high-level of the real-world experiments ran on a laptop with an 8 core i7 CPU, 8GB RAM and an NVIDIA GeForce GT 650M graphics card.

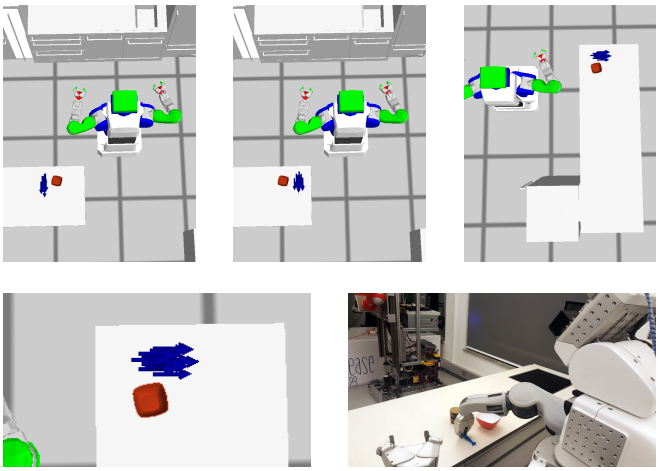

Fig. 7. Plot of all placement poses for a spoon that the human used (visualized as blue arrows), mapped onto robot's environment and reference bowl object (the red object on the table): (top left) *3Var* VR data batch was used in a similar to the real world kitchen, (top middle) *LeftSide* VR data batch was used, (top right) *LeftSide* data was used in the modified robot kitchen, (bottom left) zoomed in version of the plot, (bottom right) PR2 setting a table through *LeftSide* data.

Surprisingly, a small amount of data recorded in VR was already enough to ensure successful motion parameterizations on the robot (60 table setting episodes).

From Table IV we can see that our system often outperforms the heuristics baseline in execution success rates but encounters more intermediate recoverable failures. Action executions are overall successful, as the recoverable failures are handled by our failure handling strategies, which ask for the next answer to the same query in case of an intermediate failure. Please note that our heuristics have been improved over the years to better fit our environment and robots.

VIII. Conclusion, Discussion and Future Work

In this paper, we presented a pipeline for learning motion parameterizations of fetch and place tasks for robots, from observing humans in VR. We explained how data from VR can be transferred onto the robot and its environment using data transfer rules and how general motion parameter models can be learned based on the data. We showed that the data scales towards different environments and robot platforms and produces successful execution of a simple table setting task on different robots in simulation and real world.

Our data transfer rules have been carefully designed by hand and they are currently quite complex. Using machine learning approaches, especially data-driven deep learning models, we could simplify those rules. For example, the pose of the spoon in the table setting context is currently calculated with linear algebra formulas, based on a large amount of parameters, such as supporting surface pose and dimension, human relative pose, the pose of the reference bowl object and the robot pose. This could be simplified, in future work, by training a CNN on visual features from images of different table setting arrangements.

An important consideration for any imitation learning framework is the problem of transferring knowledge between agents with different bodies. We approach this problem by executing the plan in a simulation environment to validate the inferred motion parameterization before executing them, and by retrying the plan with the next inferred motion parameter value on failure. Naturally, the closer the robot hardware is to a human, the higher are the assumed task execution success rates and retry counts. It is unknown if our system would work on a non-anthropomorphic robot design, e.g., a quadruped. We have shown, however, that the system does scale towards different robot platforms, by performing experiments on two different robots. Our approach can be further improved by learning complete black-box mapping models between the human and robot bodies.

From the seven motion parameterizations that we have considered, we had to abstract away onto a symbolic level in only one of them: as grasping in our system is implemented as simple rigid attachments and not using dynamics and friction simulation, the degree of realism was not enough to learn exact grasping poses from the VR data. Physically stable force-based grasping is difficult to implement in a physics engine, as the hands and the object they interact with are typically represented as rigid bodies, which makes contacts rather unstable. Additionally, the dexterity of the human hand allows for rather impressive grasps, e.g., holding multiple objects in one hand. Replicating this on a robot is very difficult, especially if it has only two fingers. Thus, it is uncertain if detailed knowledge of how humans grasp objects with their hands can be of much use for a robot.

Our pipeline opens up vast possibilities for learning many more motion parameters outside of the fetch and place domain, e.g., parameters of pouring, cutting, environment manipulation, etc., which we will consider in the future.

To authors' best knowledge, in the state of the art there exists no other system that can transfer symbolic and subsymbolic data from VR onto the robot with a general pipeline. There exist specialized solutions for specific tasks. In our paper we presented a general purpose pipeline that can be used for learning any motion parameterization that is realistically enough represented in the virtual environment.

Acknowledgments

This work was supported by DFG Collaborative Research Center *EASE* (CRC #1320), DFG Project *PIPE* (project number 322037152) and EU H2020 Project *REFILLS* (project ID 731590).

References

[1] S. Schaal and C. G. Atkeson, "Learning control in robotics," *IEEE Robotics Automation Magazine*, 2010.

[2] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine, "One-shot visual imitation learning via meta-learning," in *Conference on Robot Learning*, 2017, pp. 357–368.

[3] T. Welschehold, C. Dornhege, and W. Burgard, "Learning manipulation actions from human demonstrations," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[4] ——, "Learning mobile manipulation actions from human demonstrations," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[5] T. Zhang, Z. McCarthy, O. Jowl, D. Lee, X. Chen, K. Goldberg, and P. Abbeel, "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[6] T. Bates, K. Ramirez-Amaro, T. Inamura, and G. Cheng, "On-line simultaneous learning and recognition of everyday activities from virtual reality performances," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[7] J. S. Dyrstad, E. Ruud ye, A. Stahl, and J. Reidar Mathiassen, "Teaching a robot to grasp real fish by imitation learning from a human supervisor in virtual reality," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[8] A. Haidu and M. Beetz, "Automated models of human everyday activity based on game and virtual reality technology," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019. [Online]. Available: https://ai.uni-bremen.de/_media/paper/haidu19ameva.pdf

[9] A. Haidu, D. Beßler, A. K. Bozcuoglu, and M. Beetz, "Knowrob-sim game engine-enabled knowledge processing for cognition-enabled robot control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[10] 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)*, 2015.

[11] G. Kazhoyan and M. Beetz, "Programming robotic agents with action descriptions," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[12] M. Beetz, L. Mösenlechner, and M. Tenorth, "CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[13] S. Koralewski, G. Kazhoyan, and M. Beetz, "Self-specialization of general robot plans based on experience," *Robotics and Automation Letters with IROS presentation*, 2019. [Online]. Available: https://ai.uni-bremen.de/_media/paper/koralewski19specialization.pdf

[14] D. Nyga and M. Beetz, "Reasoning about Unmodelled Concepts – Incorporating Class Taxonomies in Probabilistic Relational Models," in *Arxiv.org*, 2015, preprint. [Online]. Available: http://arxiv.org/abs/1504.05411

[15] M. Richardson and P. Domingos, "Markov logic networks," *Machine learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

[16] G. Kazhoyan and M. Beetz, "Executing underspecified actions in real world based on online projection," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019. [Online]. Available: https://ai.uni-bremen.de/_media/paper/kazhoyan19projection.pdf