2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Learning Motion Parameterizations of Mobile Pick and Place Actions 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 data acquired from observing humans in virtual environments onto robots acting in the real world, and adapting the data accordingly to achieve successful task execution. We demonstrate our pipeline by inferring seven different symbolic and subsymbolic motion parameters of mobile pick and place actions, which allows the robot to set 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. 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 in the given environment, 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. The advantage of the latter is that humans utilize intuitive physics and commonsense reasoning to correctly parameterize their motions, so their parameters are highly optimized. The disadvantage is that humans are far superior to robots in mechanical design such that it is difficult to execute motion parameters of humans on robot bodies.

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 real-world human motion tracking, VR systems allow to easily vary the environment and task scenarios, provide highly accurate ground truth data and give access to the underlying world physics. The latter means that supporting relations (the cup moves with the tray), visibility (the pot blocks the view), furniture states (the drawer is open) and force-contact events (the hand touched the cup, the cup lost contact with the tray) can be directly read out from the physics engine.

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

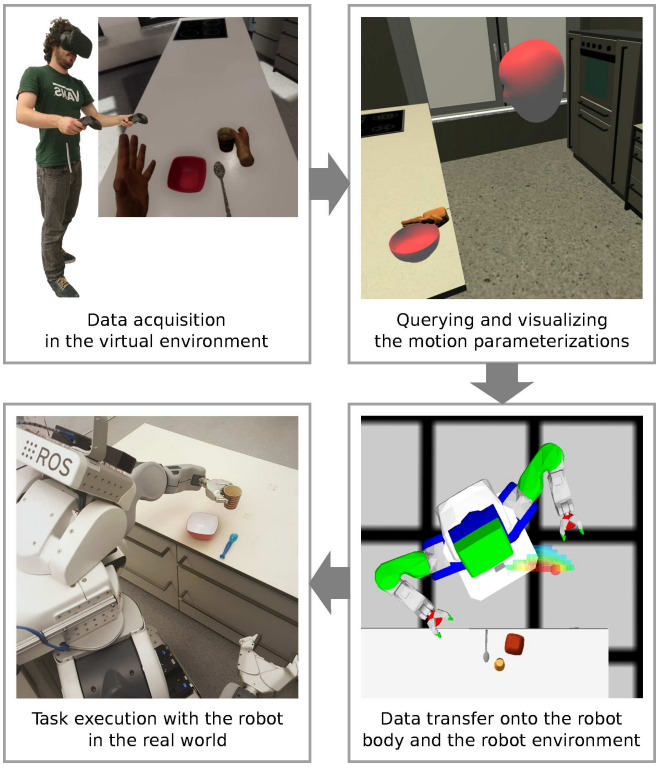

Fig. 1. Process pipeline of the presented approach: humans perform experiments in VR, the symbolic and subsymbolic observations are logged, motion parameter data are transferred onto the robot in a plan projection simulator and the learned parameters are used on the real robot for its tasks.

Given the possibility for automatic knowledge extraction from VR data, the robot can learn to parameterize its motions by imitating humans. 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 the correspondence problem between the observed actions and robot actions, and to transform the observed data into executable one. The data cannot be transferred as is, due to the differences between the virtual and the real environments and the objects in them, the physical bodies and the capabilities of robots and humans, and the contexts of tasks performed in VR and the real world. We present approaches to generalize the data such that it becomes applicable to the robot's own environment and task context. Additionally, we perform an experimental analysis of which data it is possible to transfer, and discuss what is impossible to learn from VR demonstrations with the state of the art techniques.

978-1-7281-6211-9/20/$31.00 ©2020 IEEE

An important property of our approach is that we acquire the knowledge from humans 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. Thus, our learned models are directly associated with the corresponding motion parameters, and the causal effects of good or bad parameter choices are easy to track. Factorized white box models are also advantageous when scaling the system towards other contexts, as we learn small components that can be reused.

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

- a pipeline for transferring motion parameterizations from humans in 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 seven parameters of a general mobile pick and place action, 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. We prove that the transferred data is of high quality and enables the robot to set a simple 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] use real-world human motion tracking to learn manipulation action trajectories of tasks like opening and closing drawers. They adapt the data to the robot's capabilities through a hyper-graph optimization algorithm. The data generalizes towards the robot body 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 VR. Moreover, we learn a full set of parameters, as many motion parameterizations of a certain task are interdependent (see Section IV).

Zhang et al. [4] 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 acquired 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 provide 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.

In the VirtualHome [5] and RoboTHOR [6] projects, a human avatar is controlled in a virtual environment to perform everyday tasks. The user interaction in these projects is done with a traditional keyboard and mouse, the manipulation actions are less realistic, and the applicability of the collected data to robots performing manipulation tasks in the real world is yet to be demonstrated.

An example of robots learning from human experiments in VR is [7] 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. [8] utilize VR to teach a robot to grasp real fish by observing humans perform fish grasping actions in VR. The acquired data is then used to generate a large amount of synthetic data for training a DNN. In [8] the humans control a robot gripper in VR, such that the complication of mapping between the two bodies does not arise.

In our paper, we do not concentrate as much on the learning algorithms themselves as on the general pipeline that can learn any motion parameter, 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 pipeline is not limited to these algorithms and can be combined with any other state of the art learning framework.

III. Acquiring Data by Observing Humans in VR

In this section, we outline the knowledge acquisition pipeline that is used to collect robot understandable data from observing humans in VR. We give a short overview of how the data are generated, stored and accessed. The data acquisition pipeline has been introduced in our previous work [9], thus, this section gives only a general overview.

Fig. 2. Collecting symbolic and subsymbolic data in VR

To interact with the virtual world we use an off-the-shelf VR headset with hand tracking controllers. 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: every entity from the virtual world contains a mapping to its corresponding class in the

robot's ontology [10]. Thus, every time an action (such as grasping) or a physics event (such as contact between objects) is logged, the ontological types of the objects that were participating in the logged event are also known (e.g., an object of ontological type *cup* has just been grasped).

During task execution the data is saved in two places: (1) the low-level high-frequency subsymbolic data are stored in a MongoDB database, which allows us to re-create the complete state of the world at any given timestamp; (2) the high-level symbolic data, as well as the semantic representation of the environment, are stored in an OWL ontological format.

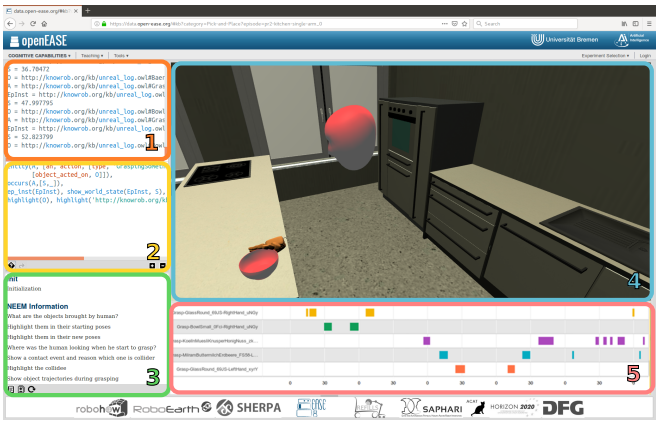

Fig. 3. openEASE web interface with visualization of data acquired in VR, where: (1) prints the results of the executed query; (2) is the input panel for Prolog queries; (3) shows predefined queries stated in natural language; (4) is a 3D rendering of the results of the query; and (5) is an interactive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

To access the data, the robot queries the knowledge base in the Prolog programming language. Humans can inspect the collected data through openEASE[1] [11], which is a web interface for answering queries with the possibility of graphical visualization. Figure 3 shows the visualization of an example query from the table setting scenario.

IV. Motion Parameters of Mobile Pick and Place

In this section, we explain the robot control programs, which we call *plans*, that command the robot during action execution. In our system, the plans are written with the so-called *action descriptions* [12], which are abstract underspecified descriptions of the actions the robot needs to execute. For example, the plan can contain the following command:

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

which describes the action of fetching an object of ontological type *cup* that has already been found in the environment. When execution reaches the command to perform an underspecified action description, reasoning queries are issued 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 (1) the arm, with which to grasp, if the robot has multiple arms, (2) the base location where to stand, such that the robot can successfully reach the object with the given arm, (3) 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, hot) 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 task, commonsense and naive physics knowledge instead of programming it directly into the robot.

Our general mobile pick and place plan is designed as a hierarchical structure. Thus, the highest-level action 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. We infer all the seven parameters in this paper.

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 ACTION, WHICH ARE LEARNED FROM OBSERVING HUMANS IN VR.

The *searching* action looks in the environment for an object that fits 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 its exact pose is known. *base-location-for-grasping* is the pose for the robot to stand, such that it is able to reach the object and the trajectory does not result in a collision. The *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 predefined 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.

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

For the *delivering* action 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. *Base-location-for-placing* is the pose of the robot base, from which *object-placement-location* is reachable. These two parameters are interdependent.

An important property of our plans is that they contain failure handling strategies. This implies that the inferred motion parameters of the plans do not have to be always correct. Instead, when a parameter results in a failure, another parameter from the list of suggestions from the (VR-based) knowledge base is picked, and the plan is retried with this new solution. As long as a suitable parameter value is sampled eventually before the retry counter is exhausted, the robot is able to execute the action successfully. 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 VR-based 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 [13].

V. Transferring Human Data onto the Robot and Its Environment

In this section, we explain how the seven parameters from Table I are inferred from the data acquired in VR, utilizing the semantic annotations in the logs.

When searching for an object, to infer *object-likely-location* we use the location of the object in the virtual environment relative to its supporting surface. It is the place, where the human put the object at the end of the last pick and place action. The object in the robot's plan is described by its ontological class, e.g., in *(an action (type searching) (object (an object (type bowl))))* we are searching for any object of ontological class *bowl*. Thus, our first constraint is that the object from the VR data is of class *bowl* or any of its subclasses. The corresponding data lookup query is:

```
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, GraspStart, GraspTime),
type(Event, 'GraspingSomething'),
property(Event, 'objectActedOn', Object),
```

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, i.e. while the object was standing on the supporting surface:

```
time_between(GraspStart, 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.

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, GraspStart, ObjectPose),
actor_pose(Episode, Surface, GraspStart, SurfPose).
```

The pose of the object in the robot's environment is then calculated with simple coordinate frame transformations:

$$ {}_{map}T_{obj} = {}_{map}T_{surface} *_{surface'} T_{obj'} = {}_{map}T_{surface} *_{map'} T^{-1}_{surface'} *_{map'} T_{obj'} $$

where ${}_aT_b$ is the transformation matrix that transforms poses in the b coordinate frame into poses in frame a, map is the global coordinate frame of robot's environment, map' is the origin of the virtual environment, obj is the frame of the object from the real world, obj' is the virtual environment coordinate of an object of the same ontological type as obj, $surface$ is the frame of the supporting surface from the real world and $surface'$ is its equivalent in VR.

The other parameters are inferred similarly. *Base-location-for-perceiving* is inferred from the pose of the human relative to the object 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, we add a constant offset 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.

Base-location-for-grasping and *base-location-for-placing* are inferred similarly. The symbolic *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 Mobile Pick 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 our previous work [14] we have used robot's own experience data to learn motion parameters of mobile pick and place. The training data was generated using heuristics to minimize the search space. However, the heuristics had to be handcrafted for each parameter. In this paper, the heuristics have been replaced by VR-based inference, such that we use the human data as seed for learning. We retrained the statistical models from [14] based on the data acquired from the robot acting in simulation with VR-based inference.

To infer grasping poses and the corresponding robot base locations, which are expected to lead to a successful fetching action, we utilize the following probability distribution:

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

where

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

Here, 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, we apply linear algebra, which allows us to determine it with a probability of 1.0.

We use Fuzzy Markov logic networks (FUZZY-MLN) [15] as a statistical model, to infer the discrete grasping pose GP, which with highest probability results in the robot grasping the object. The advantages of MLNs are that they can represent complex relations and that they are white box models, which are easy to interpret and allow to understand the causal relationships between the learned features.

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 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 when fetching a bowl and placing a cup.

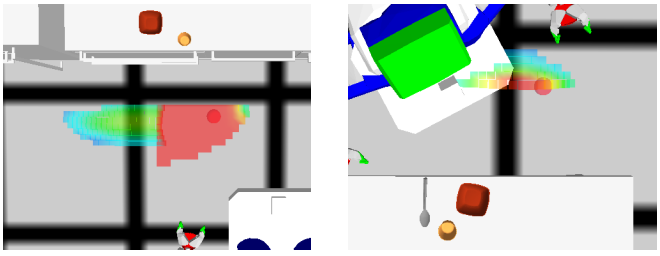

Fig. 4. Distribution for a robot base pose to (left) grasp a bowl with the left arm and (right) place a cup with the right arm, visualized as heat maps.

VII. Experimental Analysis

In this section, we empirically investigate the effectiveness of using the data from humans performing tasks in VR, for executing mobile pick and place tasks on a real robot. Specifically, we investigate the questions listed below.

1) Is it possible to successfully execute a simple table setting task on a robot by inferring motion parameters based on data from humans acting in VR?
2) How does VR-based motion parameter inference compare to the hand-crafted heuristics baseline?
3) How does the quality and quantity of the VR data affect robot performance?
4) Does the system scale to changes in the 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	Num. table setting episodes	Virtual environment	Human handedness
LikeReal	90	1 kitchen, similar to real	Right
3Var	30 * 3	3 kitchen variations	Right
3VarThird	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 *LikeReal* 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 modeled three different kitchen environments, where the scale, position and orientation of the furniture pieces have been greatly varied, creating environments as different from the real one as possible, while at the same time ensuring that the resulting arrangement is not an unrealistic human household (see Figure 5).

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 the same as the real one.

The *3VarThird* batch of collected data is simply a subset of the *3Var* batch, for evaluating quantitative effects of data on robot performance. In the *LeftSide* 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.

When collecting the data in VR, accounting for as much variation as possible is important. This includes varying the position and orientation of the objects on the surfaces, the relative pose of the human w.r.t. the objects, alternating between the different grasp poses and arms, etc.

B. Experiments

To answer the above-listed questions, we executed the table setting task in different variations, listed in Table III.

Experiment ID	Num. 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
Pr2VrSimThird	100	PR2	Simulated, similar to real	VR	3VarThird
Pr2HrSimMod	100	PR2	Simulated, modified	Heur.	–
Pr2VrSimMod	100	PR2	Simulated, modified	VR	3Var
BoxyHrSim	100	Boxy	Simulated, similar to real	Heur.	–
BoxyHrSimMod	100	Boxy	Simulated, modified	Heur.	–
BoxyVrSim	100	Boxy	Simulated, similar to real	VR	3Var
Pr2HrReal	10	PR2	Real	Heur.	–
Pr2VrRealLikeReal	10	PR2	Real	VR	LikeReal
Pr2VrReal	10	PR2	Real	VR	3Var
Pr2VrSimLeft	10	PR2	Simulated, similar to real	VR	LeftSide
Pr2VrSimModLeft	10	PR2	Simulated, modified	VR	LeftSide
Pr2VrRealLeft	3	PR2	Real	VR	LeftSide

TABLE III

Execution variations of table setting experiments.

Experiments were executed on a real robot as well as in a fast simulation environment [16], randomly varying the position and orientation of the objects' initial locations on the sink counter. In order to show that the system can be applied to different robot bodies, we used two robots – PR2 and Boxy – which are mobile manipulation platforms with two arms. Boxy is built from completely different hardware components than PR2, has other dimensions and a smaller arm reachability area (see Figure 6 (left)). To show that the system scales towards changes in the 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)). 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 take into account the preferences of the human, who collected the data in VR (see Figure 7).

Experiment ID	Success Rate (%)	Success rate per object (%)			Avg. num. non-recoverable fail.			Avg. num. recoverable failures			Execution time (s)
		Bowl	Cup	Spoon	Search	Fetch	Deliver	Percept.	Nav.	Manip.	
Pr2HrSim	89	98.5	89	100	0	0.02	0.11	12.44	6.45	47.24	39.75
Pr2VrSimLikeReal	76	98	84	94	0	0.1	0.14	10.36	321.32	19.72	49.76
Pr2VrSim	71	94	81	92	0	0.24	0.09	9.66	408.14	43.6	69.28
Pr2VrSimThird	33	83	77	52	0.02	0.7	0.15	9.79	1000.46	46.56	45.75
Pr2HrSimMod	74	88	93	91	0	0.28	0	10.12	19.31	92.28	60.1
Pr2VrSimMod	70	90	90	85	0	0.35	0	9.5	715.2	28.5	131.49
BoxyHrSim	87	99.5	88	99.5	0	0.01	0.12	9.81	21.61	70.05	64.03
BoxyHrSimMod	48	84	89	74	0	0.52	0.01	10.44	621.95	133.97	87.55
BoxyVrSim	80	83	72	90	0	0.17	0.5	4.33	254.2	70.83	147.61
Pr2HrReal	50	60	80	90	0	0.5	0.2	11	14.9	61.7	1132.66
Pr2VrRealLikeReal	90	100	90	100	0	0	0.1	4.2	203.1	53.5	1272.02
Pr2VrReal	90	100	90	100	0	0.1	0	1.56	259.4	34.8	1054.33

TABLE IV

SUCCESS RATES, NUMBER OF FAILURES AND EXECUTION TIME OF EXPERIMENTS FROM TABLE III, AVERAGED OVER THE NUMBER OF EXECUTIONS.

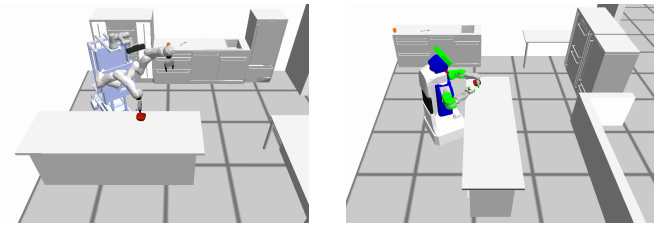

Fig. 6. Executing the table setting scenario (left) in a similar to real-world kitchen on a different robot and (right) in a different-from-real environment.

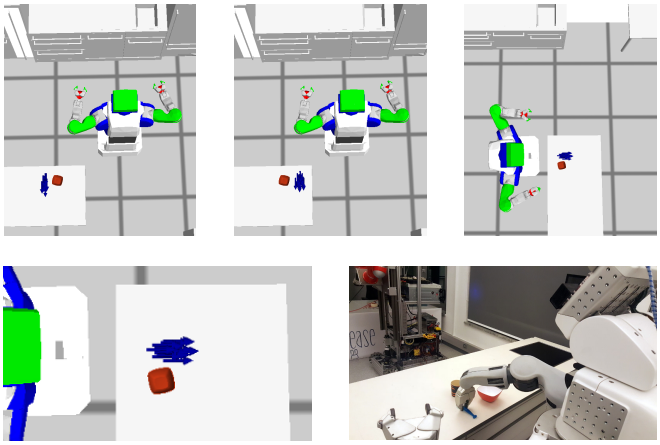

Fig. 7. Plot of all placement poses for a spoon that the human used, visualized as blue arrows, mapped onto the robot's environment and the reference bowl object (the red object on the table): (top left) *3Var* VR data batch was used in a kitchen, similar to the real world, (top middle) *LeftSide* VR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

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. All the experiments in Table III, which have *Heur.* as the inference engine, are our baseline.

Table IV shows the results of the experiments, averaged over the number of table setting executions per experiment. The execution time is the average duration of one table setting run, only considering the successful ones. The number of failures is averaged over the number of executions (which turns natural numbers into reals). Recoverable failures are those that can be corrected by retrying the plan with another parameter. The experiments, where the number of recoverable failures is high, are those where there are fewer suitable motion parameters in the VR data (e.g., in *Pr2VrSimThird* only a third of the data was used), so the robot has to try out a very large amount of parameters until a good one is found. The non-recoverable failures are thrown, when the robot gives up on transporting the given object.

By looking at execution success rates, we can conclude that our system could successfully learn the seven motion parameters of a transporting action from Table I. We can see that our system often comes close to our baseline of handcrafted heuristics in execution success rates, although it encounters more intermediate recoverable failures.

The heuristics-based experiments have much higher success rates in the kitchen, similar to the real one, than in the modified kitchen: 89% vs 75% for PR2 and 87% vs 48% for Boxy. The modified kitchen is more challenging than the original one, as the objects are often spawned in the corner of the room, where the robot base poses are very restricted.

In the similar to the real-world kitchen, the VR-based approach has 71% success rate, versus the 89% of the heuristics. We consider 71% a satisfactory success rate, given that heuristics require a considerable amount of manual work, whereas the data collection in VR can be collected by non-experts and can be crowd-sourced.

The most important experiment is, we argue, the one in the modified robot kitchen. Heuristics performed with 74% success rate there, which was 15% worse than in the original kitchen. The VR-based approach resulted in 71% success, which is only 1% worse than in the original kitchen. Thus, we conclude that the VR-based approach generalizes well towards different environments, even the challenging ones.

The real-world experiments only contained 10 execution runs, which we consider insufficient to make statistically significant claims but important as a proof of concept.

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 in the real world. Due to the use of the ontology, the system can also scale to novel objects, as long as they belong to an ontological subclass of a known object from the virtual environment.

Our data transfer rules have been carefully designed by hand. Using machine learning approaches, especially data-driven deep learning models, we could simplify these 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 parameterizations 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 success rates. 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 scales towards novel robot platforms by performing experiments on two different robots. Our approach can be further improved by learning black-box mapping models between the human and the 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 objects they interact with are typically represented as rigid bodies, which makes contacts rather unstable. Additionally, the dexterity of the human hand is far superior to that of a robot, and allows for 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 the 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 mobile manipulation actions from human demonstrations," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[4] 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.

[5] X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba, "VirtualHome: Simulating household activities via programs," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[6] M. Deitke, W. Han, A. Herrasti, A. Kembhavi, E. Kolve, R. Mottaghi, J. Salvador, D. Schwenk, E. VanderBilt, M. Wallingford, L. Weihs, M. Yatskar, and A. Farhadi, "RoboTHOR: An open simulation-to-real embodied ai platform," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] S. Koralewski, G. Kazhoyan, and M. Beetz, "Self-specialization of general robot plans based on experience," *Robotics and Automation Letters with IROS presentation*, 2019.

[15] 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

[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.