Synthesizing Robot Manipulation Programs from a Single Observed Human Demonstration

Justin Huang[1], Dieter Fox[2], and Maya Cakmak[2]

***Abstract*— Programming by Demonstration (PbD) lets users with little technical background program a wide variety of manipulation tasks for robots, but it should be as intuitive as possible for users while requiring as little time as possible. In this paper, we present a Programming by Demonstration system that synthesizes manipulation programs from a single observed demonstration, allowing users to program new tasks for a robot simply by performing the task once themselves. A human-in-the-loop interface helps users make corrections to the perceptual state as needed. We introduce *Object Interaction Programs* as a representation of multi-object, bimanual manipulation tasks and present algorithms for extracting programs from observed demonstrations and transferring programs to a robot to perform the task in a new scene. We demonstrate the expressivity and generalizability of our approach through an evaluation on a benchmark of complex tasks.**

I. Introduction

Many everyday tasks in human environments, from preparing a meal to cleaning surfaces with a tool, require complex manipulation capabilities. Developing universal manipulation capabilities that work for every possible task is extremely challenging. Instead, researchers are putting effort to developing ways in which people can quickly and easily program new robot capabilities. Programming by Demonstration (PbD) is a popular approach that enables people without a technical robotics background to program new capabilities on the robot by demonstrating the task to the robot in some manner. Three decades of research on the topic has resulted in a wide variety of approaches. Despite these advances, work in this area has been limited in terms of the level of user training and involvement (*e.g.*, when using kinesthetic guidance or teach pendants) and how many demonstrations are required (typically more than one).

An ideal approach is to have the robot observe a human demonstrate a task once, and to have the robot subsequently be able to perform the same task. However, this is hard for two reasons. First, state of the art perception systems still have trouble accurately tracking object poses, especially for small objects that can be occluded by the demonstrator's hand. Second, it can be difficult for the robot to correctly infer the intent of the demonstration, especially when only one demonstration is provided. In this paper, we present a system for programming a robot to perform complex object manipulation tasks by observing a single RGBD video demonstration. To address the perception problem, we provide users with a human-in-the-loop interface that combines an automatic object tracker with human supervision. Rather than have the robot try to understand the intent of the user, our system converts the demonstration into a program using a small set of rules that are easy for users to understand, putting users in control of the programming process.

We also present a representation of a manipulation task called an *Object Interaction Program*. This representation is robot-independent and allows the task to be repeated with varying starting configurations of objects. In contrast to simple PbD systems that simply record and replay taught poses, our approach uses several unique modeling and planning techniques to adapt its grasps and trajectories to maximize task success. This paper presents our system along with the novel representation and algorithms that we developed. Our work contributes:

- An expressive representation of complex manipulation tasks called *Object Interaction Programs*.
- New algorithms for (i) synthesizing task programs from a single observed demonstration, and (ii) transferring programs to the robot in new scenarios for execution.
- Motion capture tools for semi-automatic annotation of rich kinematic data from RGBD video without the use of fiducial markers.

Our approach was implemented on a PR2 robot, and the expressivity and robustness of our approach was evaluated through a benchmark of complex, multi-object, bimanual tasks.

II. Related Work

Our work contributes to a long line of research on robot Programming by Demonstration (PbD), also referred to as Learning from Demonstration [1], [2]. In the following we situate and differentiate our work within the landscape of PbD research in terms of the demonstration method, the types of tasks being learned and the learned representation, as well as the process of going from a demonstration to an execution. The goal of our research also aligns directly with prior work within the HRI community on making PbD more intuitive non expert users [3]–[7] and making it easier for them to program robots in different ways [8]–[10].

The influential survey by Argall *et al.* identified demonstration method as one of the key design choices in a PbD system. They define *embodiment mapping* to differentiate between demonstrating a task by making the robot do the task (*e.g.* through teleoperation [11], verbal commands [12], or kinesthetic guidance [13]) versus having the human do the task. Our work falls under the second category, which they refer to as imitation. Demonstration methods are further

[1] Jet Propulsion Laboratory, Pasadena, CA 91109, USA. Work done while affiliated with the University of Washington.

[2] Paul G. Allen School for Computer Science and Engineering, University of Washington. Seattle, WA 98195, USA.

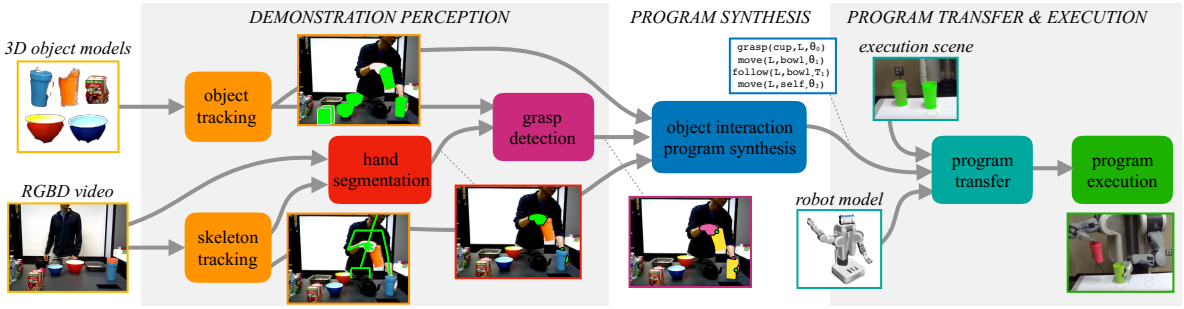

Fig. 1: Overview of the Programming by Demonstration system developed in this work.

categorized based on *record mapping* to differentiate between demonstrations perceived through sensors on the human (1st person perspective) versus externally through sensors on the robot (3rd person perspective). Our work falls in the second category as we use an RGBD camera on the robot to perceive the demonstration. Recently, researchers have also collected demonstrations in virtual reality environments [14], [15], which are harder to set up but provide perfect information.

Inferring human states and actions from sensor observations is a challenging problem by itself. Earlier work in this same category mostly relied on simplifications such as attaching markers to the person [16], [17], using motion capture [18], [19], or equipping the demonstrator with wearable sensors like data gloves [11], [20]. Our system does not require the use of markers for motion capture, leveraging recent advances in computer vision research. Others have explored learning a mapping from the robot's view to the user's view and learn a low level policy to replicate the demonstrated visual change [21], [22].

The way PbD systems represent tasks can be broadly divided into those that create short-term arm trajectories (*e.g.*, pouring a cup or opening a cupboard) and those that build higher-level actions (*e.g.*, changing a tire) using a catalog of low-level skills. In the first category, researchers have used statistical methods like Gaussian Mixture Models [23] or Hidden Markov Models [24] to find commonalities between multiple demonstrations of the same task. Dynamic Movement Primitives (DMPs) represent trajectories as an attractor landscape that can be parameterized by a goal configuration [25]. Recently, researchers have studied deep reinforcement learning algorithms that, given video demonstrations of a task, learn policies that directly map image observations to low-level robot controls [15], [22], [26], [27]. In the second category, learned high-level task representations include finite state machines [28], hierarchical task networks [29], and knowledge-based ontologies [30]. Work in this area also includes goal-based representations, in which the robot makes plans to achieve a goal state without the need for human demonstration [31]. While the approaches described above are powerful, they lack explainability and are difficult for users to modify if the behavior is not quite right. To enable users to easily program custom robot behaviors, we designed our task representation to be an easy-to-understand, rule-based approach.

A problem with most of the approaches described above is that they require multiple demonstrations or hundreds of videos for the system to acquire the skill. For novice users, it may not be intuitive or convenient to provide multiple demonstrations. More recently researchers have placed more emphasis on learning from a single (as in our case) or very few demonstrations, leveraging training with prior knowledge [27] task structure [29], augmenting the demonstration with different interactions such as asking questions [3], [32], interacting with visualizations [33], or using the demonstration to bootstrap a self exploration process [34].

III. System Overview

Our system has roughly three parts, illustrated in Figure 1 and described in more detail in the sections below[1].

Perception: First, we extract relevant information from an RGBD video of a demonstration by tracking the object poses and the demonstrator's hands. This step is mostly automated, but we allow users to make manual corrections as needed.

Program synthesis: Next, we convert the extracted perceptual information into an *object interaction program* by first segmenting the demonstration based on certain detected events, then assigning each segment to a specific action. The steps described so far can be performed offline.

Program transfer & execution: Lastly the program is transferred to a specific robot and execution scene at runtime. First, objects used in the task are located in the scene. We then plan object grasps, taking into account the type of grasp that is needed, the possibility of collisions, and more. Once grasps are selected, all object trajectories observed in the perception step are converted to end-effector trajectories based on the positions of the objects in the current scene. Finally, task-space trajectories are converted to joint-space trajectories and executed on the robot.

IV. Perception of Human Demonstrations

A. Object tracking

Replicating complex object interaction tasks requires access to the full 6-DOF trajectories of the objects. While robust tracking of object poses in cluttered scenes is still an active area of research, recent methods perform reasonably well in tabletop task settings. In this paper, we use a particle

[1]The open-source software for the system is available at `https://github.com/jstnhuang/task_perception`.

filter-based tracker by Wüthrich *et al.* [35]. This method requires point-cloud models of objects to be tracked. The initial poses of these objects can be determined automatically as we describe in Section VI-A; however, we let users manually align the object models to their starting poses. Given an RGBD video of K frames, the output of the object tracker with N objects is the sequence of $SE(3)$ poses θ_{ij} of each object in each frame, $O = \{\theta_{ij} : i = 1..N; j = 1..K\}$. We provide users with an interface to visualize the output of the tracker for each frame. If the tracker drifts away from the correct answer, the user can adjust the pose, which reinitializes the tracker to the adjusted pose.

B. Skeleton tracking

To track the demonstrator's movements in 3D, we use a human body tracker by Walsman et al. [36]. To initialize the body tracker, the demonstrator needs to stand with arms held out to the side. Our approach does not need the full skeleton information—just an estimate for the pose of the demonstrator's wrists. Formally, given an RGBD video of K frames, we extract the right and left wrist poses for each frame, $W = \{\theta_j^R, \theta_j^L; j = 1..K\}$. We use this information later for determining grasp and ungrasp events and to help plan future grasps of the same object.

C. Hand segmentation

An important component of the system is to track the demonstrator's hands with higher accuracy and precision than the skeleton tracker can provide. This is especially useful to tell when the demonstrator grasps or ungrasps an object. To that end, we train a deep neural network, based on a single-frame model derived from Xiang *et al.* [37], that labels each pixel of the RGBD image as hand or not hand.

To train the model, we generated a dataset of 23,928 frames of RGBD video recorded with an ASUS Xtion Pro RGBD camera. These videos involved one of two individuals moving their hands and manipulating objects in an office setting. To label the data, we mounted a thermal camera next to the RGBD camera. In the thermal camera image, exposed parts of the person's body such as the hands, arms, and face show up as brighter pixels. Additionally, the skeleton tracker helped isolate the hands from other parts of the body. A custom annotation interface was used to step through the videos to inspect the automatically detected labels and make corrections.

To assess the performance of the hand tracker, we split our dataset randomly with 80% in a training set and 20% in a validation set. We measured the intersection over union (IOU), which is the same as accuracy but excluding all true negative pixels[2]. Our model labeled hand pixels with an IOU score of 66.46%. We found that the output tended to lose fine details like fingers, likely due to upsampling operations that take place in the neural network architecture. Previous work has shown that this issue can be ameliorated by adding skip layers to the network [38]. We found that by simply applying an erosion operation with a 3x3 kernel, the IOU of hand segmentation improved to 72.64%.

[2]We use IOU because our dataset contains heavy class imbalance. Only 0.78% of pixels are hand pixels, meaning a trivial algorithm that predicts all negatives would have an accuracy of 99.22%.

D. Grasp detection

We combine object tracking information with the hand segmentation to determine when the demonstrator's hand makes or loses contact with objects in the scene. We first project the hand pixels into 3D coordinates using the depth of the RGBD image. At each frame of the demonstration, we consider a 3D hand pixel to be "touching" an object if its distance to the object is below a threshold. Then, we determine how many hand pixels are touching each object. If the number is above a threshold for one of the objects, the hand is considered in contact with that object. Hence, for a demonstration of K frames, we obtain $C = \{c_j^R, c_j^L; j = 1..K\}$ where $c_j^{R/L} = i \in \{1..N\}$, the index of the object that the hand is in contact with, and 0 if there is no contact.

V. Program Synthesis

The previous step generated, for each video frame, the poses of all the objects in the task, as well as grasp and ungrasp events and the poses of the demonstrator's wrist. This section describes how that information is converted into a program that encodes the semantics of the task.

A. Object Interaction Programs

Our work focuses on complex object manipulation tasks between objects such as arranging them, disassembling them, or applying a tool to an object. These tasks involve grasping, transporting objects in the workspace closer to one another, and moving them relative to one another. We propose to represent such tasks as a sequence of the following primitive actions:

- `grasp(object, arm-side, ee-pose)`
- `move(arm-side, ref-object, pose)`
- `follow(arm-side, ref-object, trajectory)`
- `ungrasp(arm-side)`

The `grasp` action involves grasping the object from a preferred end-effector pose with the specified arm. The exact grasp pose of the end effector might differ from the specified one depending on the robot and the execution scene. The `move` action involves moving a previously grasped object to a target pose relative to a reference object. If the movement is not relative to another object, then the target pose is relative to the initial pose of the object being moved. The `follow` action involves following a dense object trajectory of arbitrary length relative to a reference object. The `ungrasp` action involves simply opening the gripper and moving away from the object. The parameters of the actions have the following types: `object` and `ref-object` are unique object type identifiers; `arm-side` $\in \{R, L\}$; `pose`, `ee-pose` $\in SE(3)$; and `trajectory` $\in SE(3)^M$ for a trajectory segment with M frames.

An *object interaction program* (OIP) is a sequence of actions instantiated parameters. OIPs capture task information in terms of object poses and trajectories, independent of the

agent (human or different robots) performing the actions to manipulate objects. Next, we describe how we go from a single demonstration to an OIP.

B. Program synthesis

The first step in synthesizing an OIP from an observed demonstration is segmenting the demonstration. We identify the set of indexes of all segmentation points, S, as the union of frames where:

- A grasp or ungrasp event happens,
- Two objects start or stop interacting, *i.e.*, the distance between the objects exceeds or falls below a threshold.

Next, we assign each segment to a pair of OIP actions, one for each arm of the robot, based on the demonstration. If there are $|S|$ segmentation points, then $2*(|S|+1)$ segments need to be assigned an action. Each segment is associated with an OIP action using easy to understand and explainable rules.

First, if a segment ends at frame j with a grasp event ($c_j > 0$ and $c_{j-1} = 0$), we add a `grasp` action, storing the ID of the object, whether the grasp was with the right or left arm, and the pose of the demonstrator's wrist at the time of the grasp. If demonstrator moves the object close to another object (*i.e.*, an object interaction), we add a `move` action. We store the ID of the other object in `ref-object` and the relative pose of the objects in `pose`. If the demonstrator continues to move an object in the vicinity of another object, we add a `follow` action. As with a `move` action, we store the ID of the other object. We also store the `trajectory` of the held object relative to the other object. Otherwise, if the demonstrator moves the object away from any objects, the choice of `ref-frame` is a bit ambiguous. In our implementation, we set the `ref-frame` in this case to be the object's pose at the start of the demonstration, although other choices are possible. Finally, if the segment ends with the demonstrator ungrasping an object, we add an `ungrasp` action. An example of the segmentation process is shown in Figure 2.

One special case is when two objects being held by the two arms interact with each other. To keep the programming model simple, OIPs do not support dynamic coordinate frames. This means the demonstrator is holding one of the objects with the intent of holding it still. However, the system might detect a small amount of movement for the stationary object. In these situations we detect the object that moves the least eliminate all movement by removing the `follow` actions in the corresponding segments. The stationary object serves as the landmark for the object held in the other hand.

VI. Task Transfer and Execution

OIPs represent tasks independent of the robot and particular execution scene. Performing a task represented with an OIP requires the program to be transferred to the particular robot and scene. This is akin to a high level programming language that can be compiled on different systems to perform the same tasks. Our algorithm for doing so is described below.

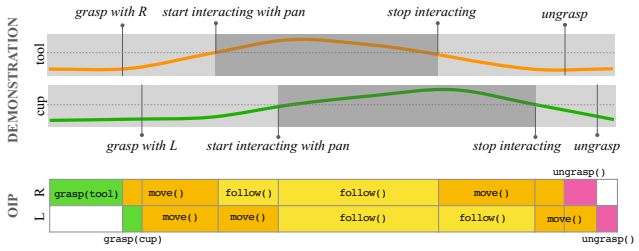

Fig. 2: (Top) Illustration of a recorded demonstration with segmentation points highlighted, and (Bottom) the corresponding OIP with each segment assigned to an action. The example corresponds to the *pour and stir* task from our evaluation.

A. Object localization and registration

The first step when running a program is to localize the objects in the scene. In some cases, there may be two identical objects in the scene that serve different purposes (*e.g.*, two cups, one of which is poured into the other). This is the object registration problem, and the robot must do both localization and registration before executing a program. Because our system is already equipped with object models and our tasks take place in a tabletop setting, we developed a simple matching approach to solves both problems. To localize the objects, we fit a plane to the tabletop surface and segment objects above the plane. We then match the objects to their models based on shape and iteratively refine their poses using the ICP algorithm [39]. If there are multiple instances of the same object, we assign the objects' roles based on closeness to the initial object poses in the original demonstration. After the robot performs the object localization, we allow the user to edit the pose as needed for certain challenging scenes.

The OIP stores the object trajectories as poses relative to other objects in the scene. Once the robot has localized the objects in the new scene, it can re-compute the desired object trajectories to be poses in the robot's coordinate frame. However, rather than simply replaying these object trajectories, our system performs several more planning steps to help ensure task success, described below.

B. Grasp selection

Grasp selection is a key component of our system, because once an object is grasped, it must accomplish all future `move` and `follow` actions without changing its grasp. As in classical grasp planning problems, finding a successful grasp on an object depends on (i) the robot's gripper, (ii) the 3D model of the object, (iii) the robot's manipulator kinematics, (iv) the pose of the object relative to the robot, and (v) clutter around the workspace. In addition, the demonstration presents two additional constraints we try to satisfy in grasp selection:

- The robot should ensure that its grasp allows it to reach all future poses of the object specified by subsequent `move` and `follow` actions.
- The robot should grasp objects as similarly as possible to the grasp demonstrated by the human.

Our grasp planning algorithm first generates candidate grasps. While alternative candidate generation methods are possible, we randomly sample points on the object and approach angles. A useful heuristic that we implemented is to change the grasp depth based on the width of the object inside the grasp. If the robot is grasping something thin, it uses a fingertip grasp, but if it is grasping something large, it uses a power grasp. This tends the correlate well with how humans grasp objects. For radially symmetric objects (such as cylinders, cups, bowls, or plates), grasps that are rotations around the symmetry axis are considered equivalent. Grasps that are unreachable or in collision with workspace obstacles or the target object are eliminated. We then compute a score for remaining candidate grasps as the weighted sum of several factors:

- *Antipodality*: The number of points whose normals are aligned with the gripper jaw.
- *Collisions*: The number of points that would be in collision with the gripper.
- *Similarity to demonstration*: The Euclidean distance between the grasp pose and the demonstrator's wrist at the time of the grasp.

For each of the top grasps, we compute another score factor: the kinematic feasibility of this grasp if it were used. We look ahead at future object trajectories and compute what the robot's gripper pose would be with the candidate grasp. We add another weighted term to the grasp score: the percentage of poses that have no *relaxed inverse kinematics* (described in Section VI-C) solution. Finally, we choose the best grasp.

Figure 3 and Figure 4 show example pairs of demonstrated human grasps and generated robot grasps in different tasks used in our evaluation.

C. Generating motion plans

Once a grasp has been selected, we compute robot end-effector poses from all the object poses in the OIP. The target `pose` of a `move` action, which is specified in the coordinate frame of the `ref-object`, is first converted into the robot's coordinate frame based on the `ref-object`'s known pose in the robot's coordinate frame. Then, the grasp pose is used to compute the end-effector pose that corresponds to the target `pose`. The same process is used for converting object-relative object trajectories into end-effector trajectories.

Unlike `follow` actions, `grasp`, `ungrasp`, and `move` actions only specify goal poses, not trajectories. To find collision free paths that reach a desired final pose, we use existing motion planning methods in the open-source *MoveIt* library[3]. We first try planning a straight-line motion for the gripper. If this plan is not feasible, we use the RRT-Connect motion planning algorithm. If this does not work, then we declare a failure to execute the program.

For `follow` actions, the end effector trajectory is converted into joint trajectories using what we call *relaxed inverse kinematics*. Relaxed inverse kinematics is the same as normal 6 degree-of-freedom inverse kinematics, but it takes the radial symmetry of certain objects into account (*e.g.* cups, bowls, cylinders). If, for a radially symmetric object, the robot fails to find an IK solution, it considers alternative poses by radially rotating the object. This allows the robot to better find IK solutions, as well as avoid collisions when its two grippers are close together.

The final step before execution is to re-time the trajectories in case the demonstrator's motions are faster than the robot's velocity limits would allow. This is done on a segment-by-segment basis, using the segments found in Section V-B. To ensure that the ordering of grasp, ungrasp, and object interactions is unchanged after retiming, we compute the slowdown factor for the arm joint that has to be slowed down the most, and then scale the other joints' velocities by the same factor.

[3]https://moveit.ros.org

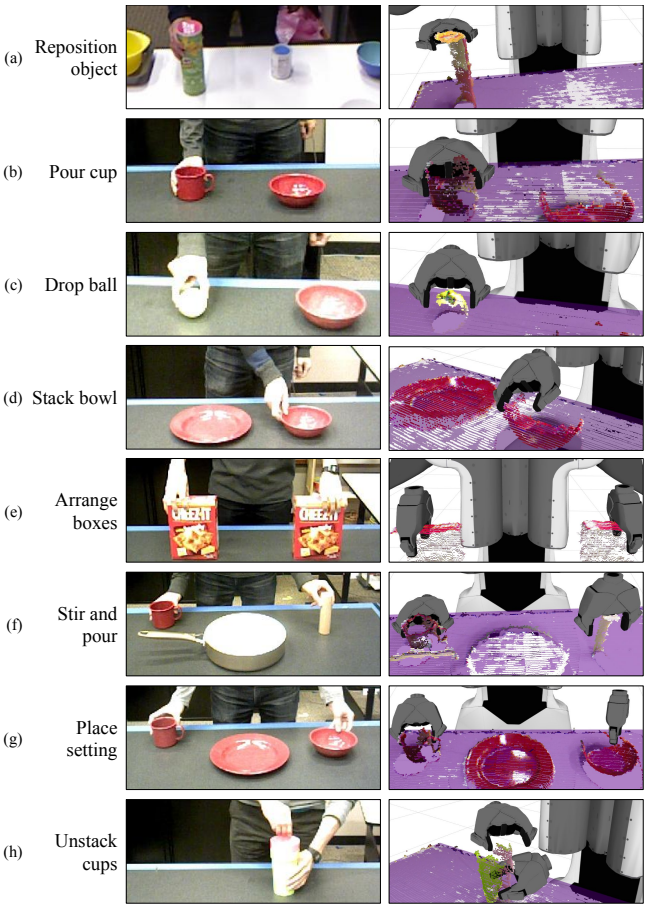

Fig. 3: Examples of grasps generated based on human demonstrations. Each row corresponds to a task described in Section VII-A. Images on the left show frames from the demonstration at the moment the demonstrator grasps the object(s). Images on the right show the grasps selected by our system.

VII. Evaluation

In our evaluation, we show that how OIPs are expressive enough to represent complex manipulation tasks with unique properties. We created a benchmark of eight manipulation tasks with different kinds of motions, including bimanual

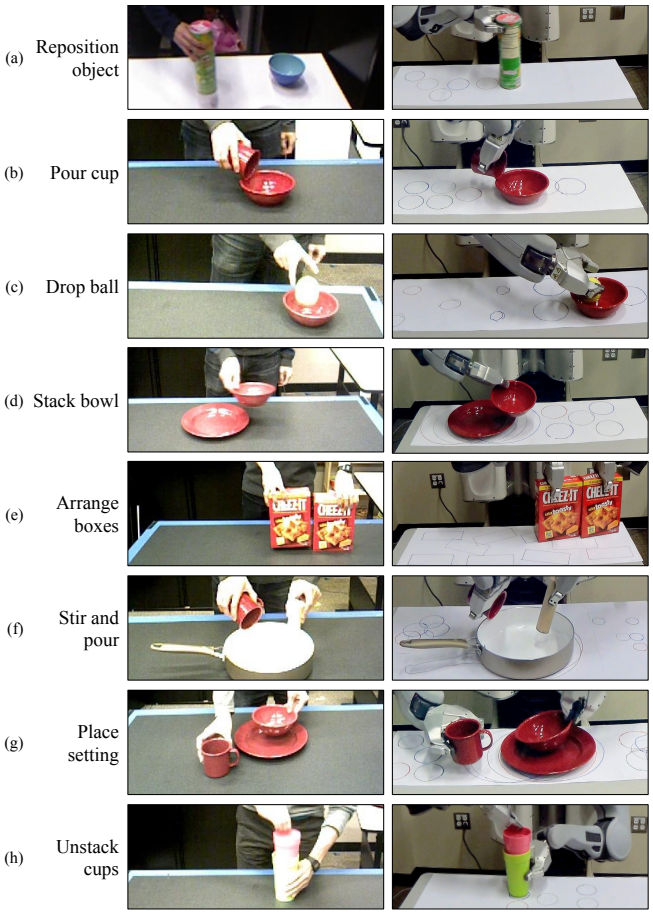

Fig. 4: Eight tasks that were programmed in the evaluation of our system. The left image of each row shows a still frame from the demonstration video and the right image shows the robot executing the task at the same point in the task. Tasks are described in detail in Section VII-A.

tasks. We also show that our programs generalize across varying starting configurations of the objects in each task.

A. Tasks

Our benchmark includes eight tasks, four that use a single arm and four that use two arms at the same time. Below, we describe each of the tasks, how they are unique, and how we defined success for each task. Figure 4 depicts these tasks.

1) Reposition object: In this task, the robot picks up a canister of chips and places it in a new location. Although simple, this task tests a special case in our system, where the reference object in a `move` action is the object itself. In other words, the generated program simply shifts the object's pose relative to its initial pose. A trial was successful if the robot successfully grasped the object and moved it from one side of the table to the other.

2) Pour a cup into a bowl: In this task, the robot picks up a cup and pours its contents into a bowl. This task demonstrates a trajectory-following action. Once the robot moves the cup near the bowl, it follows the human-demonstrated trajectory of tipping the cup into the bowl. Additionally, it shows how our system can choose a grasp based on human demonstration. In the absence of task-specific information about the cup, the robot could easily grasp the cup from above, sticking one gripper finger inside the cup. However, because the demonstrator grasped the cup from the side, we expect the grasp planner to choose a side grasp as well. A trial of this task was successful if the robot grasped the cup from the side, executed the pouring motion, and placed the cup back down on the table.

3) Drop a ball: In this task, the robot picks up a tennis ball and drops it into a bowl. There are two unique features to this task. First, because the ball is spherical, the automatic object tracker cannot correctly determine its orientation throughout the demonstration. As a result, our system must use the *relaxed inverse kinematics* capabilities described in Section VI-C. This task is also an example of an ungrasping action that is detected by dropping an object (*i.e.*, the object moves away from the person's hand rather than vice versa). A trial of this action was successful if the robot successfully picked up the ball and dropped it into the bowl.

4) Stack a bowl: For this task, the robot picks up a bowl and places it on a plate. The robot should grasp just the bowl's rim rather than place its fingers deep inside the bowl. We also used this task to illustrate how our system deals with cross-body motions in demonstrations. The bowl was placed to the demonstrator's left, but the demonstrator grasped it using their right hand. A trial of this task was successful if the robot placed the bowl on the plate.

5) Arrange boxes: For this task, the robot moves a box to place it in contact with another box. The unique feature of this task is that it requires the robot to hold one box steady with one arm while the other arm pushes the other box against it. A trial of this task was successful if the robot successfully pushed one box against the other without moving the steadied box.

6) Stir and pour: This task simulated a cooking action in which one arm pours a cup into a skillet while the other arm stirs the skillet with a tool. This task is unique because it shows the arm executing two trajectory-following actions simultaneously. The two arms also come in close proximity over the skillet in this action. Another unique feature of this task is that the robot can avoid collisions by holding the tool differently. Because the tool used in our experiments is radially symmetric, the robot can hold it in such a way that prevents its arms from colliding. A trial of this task was successful if the robot successfully grasped the two objects, executed the two motions simultaneously without its arms colliding, and placed the objects back on the table.

7) Place setting: In this task, the robot must simultaneously arrange a bowl on a plate and a cup next to the plate. The unique feature of this task is that one of the arms is doing a pick-and-place action (placing the cup next to the plate), while the other is doing a trajectory-following action (placing the bowl on the plate). A trial of this task was successful if the robot successfully grasped both the cup and the bowl and placed them in the correct locations.

8) Unstack cups: In this task, there is a stack of two plastic cups and the robot needs to separate them. Because

there is a small amount of suction between the two cups when they are stacked, one arm must be used to grasp the bottom cup while pulling the top cup with the other arm. The bottom cup must be grasped using a power grasp from the side while the top cup must be grasped from the rim without grasping the bottom cup as well. Unlike other tasks in our benchmark the two objects start close together and the robot separates them. A trial of this task was considered a success if the robot successfully separated the two cups and placed the top cup on the table.

B. Procedure

The experiments took place in an indoor laboratory with the Willow Garage PR2 robot. A demonstrator recorded video demonstrations using an ASUS Xtion PRO Live RGBD camera. All tasks were demonstrated once and executed five times. We did not re-demonstrate tasks based on the outcome of the executions. One of the authors initialized and verified the output of the perception system, making corrections when needed. The objects were placed in different starting positions for each execution.The minimum amount of translation across all objects and trials was 1.8 cm, the maximum was 26.7 cm. Program executions were observed by the experimenter and categorized as successful or failed based on the task criteria described in Section VII-A. Reasons for failures were noted.

C. Results

All tasks were successfully programmed with a single demonstration. The generated OIPs matched expected sets of actions with the correct ordering. Our task transfer and execution method successfully generated a plan in all trials except one; *i.e.* the robot was able to attempt the execution in 39 out of the 40 trials. Table I shows the success rate of the program executions across five trials for all tasks. Executions of all tasks were successful in at least four out of the five trial configurations. Four out of the eight tasks had one failure. In total, 36 out of the 40 executions were successful.

The reasons for the failed trials were noted as follows. For one trial of the *pour cup* task, the robot was unable to generate a plan to reach a bowl placed farther away than it was in the demonstration. Hence, it could not attempt an execution. The next two failures were due to errors in the motion execution. In the *stack bowl* task, the robot failed one trial by missing a grasp. This was likely due to noise in the estimated pose of the bowl, which was grasped very close to the rim and therefore had the low tolerance. In the *unstack cups* task, the robot did not get a strong enough grip on the top cup, and the top cup slipped out of its grasp.

The last failure pointed to a limitation in expressivity of our representation. In the *stir and pour* task, the robot is required to place the tool back on the table. In one trial, this placement location (which is relative to the starting pose of the tool) ended up past the edge of the table, and the tool fell to the ground. Although our system detects the table surface and knows its location and dimensions, it only uses this information to avoid colliding with the table. In an improved system, the robot could use this information to avoid dropping objects past the edge of the table. Alternatively, the table itself could be considered a landmark allowing specified poses to be relative to the table, instead of the object's initial pose. Nonetheless, this was a minor failure as the rest of the stirring and pouring action were performed successfully in the trial where the tool was dropped off the table.

Task	**# success**	**Task**	**# success**
Reposition object	5	Arrange boxes	5
Pour cup	4	Stir and pour	4
Drop ball	5	Place setting	5
Stack bowl	4	Unstack cups	4

TABLE I: Number of successful trials (out of 5) for each task in the evaluation.

We also evaluated the system's level of autonomy in the process of programming the eight tasks. For this purpose, we noted the number of video frames in which the experimenter had to make corrections to the output of the perception system. The total number of video frames in the demonstrations varied between 116-325 (M=185.75 SD=72.95). We saw that the skeleton tracking was overall very robust and required adjustments in only three of the tasks for 1-4 frames. In contrast, object tracking required corrections in five out of the eight tasks. The most problematic task was *unstack cups*, which required adjustments in 74 out of 116 frames of the demonstration. This was because the object tracker could not distinguish between the top and bottom cups while the two cups were nested. Additionally, we noticed that the object tracker had a hard time tracking small objects when occluded by the demonstrator's hand. These objects included the mug (in *pour cup*), the stirring tool (in *stir and pour*), and the tennis ball (in *drop ball*). As a result, the tasks involving those objects required adjustments in 11.6% to 31.4% of the demonstration frames.

Overall, our evaluation demonstrated both expressivity across complex task types and the ability to generalize across different task execution scenes. The perception challenge of tracking small (human hand-sized) objects proved to be too difficult to enable a fully autonomous system. However, our system adopted a practical alternative, producing working programs after some minimal user supervision.

VIII. Conclusion

We present a system that leverages recent advances in robot perception for programming complex manipulation tasks from a single execution of the task observed from a third person perspective through the robot's sensors. We propose *object interaction programs* as a robot-independent task representation that captures how objects move relative to one another as part of the task. We develop algorithms to parse a single demonstration into a program, and later transfer the program to a robot in a new scene for execution. We present a benchmarking evaluation that demonstrates expressivity across bi-manual task with a variety of objects and varying object starting positions. While the perception

challenge of tracking nested objects or small objects in the person's hand prevented our system from being fully autonomous, we provide users with tools to supervise the object tracking and initialization steps of the system. We believe that incorporating greater degrees of autonomy alongside human supervision will be an exciting area of continuing research.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation, Awards IIS-1552427 "CAREER: End-User Programming of General-Purpose Robots" and IIS-1525251 "NRI: Rich Task Perception for Programming by Demonstration."

REFERENCES

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

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

[3] M. Cakmak and A. L. Thomaz, "Designing robot learners that ask good questions," in *Proceedings of the International Conference on Human-Robot Interaction (HRI)*, 2012.

[4] B. Akgun, M. Cakmak, K. Jiang, and A. Thomaz, "Keyframe-based learning from demonstration," *(In press) Journal of Social Robotics, Special issue on Learning from Demonstration*, 2012.

[5] H. Suay, R. Toris, and S. Chernova, "A practical comparison of three robot learning from demonstration algorithms," *Intl. Journal of Social Robotics, special issue on LfD*, vol. 4, no. 4, 2012.

[6] N. Koenig, L. Takayama, and M. Matarić, "Communication and knowledge sharing in human–robot interaction and learning from demonstration," *Neural Networks*, vol. 23, no. 8, 2010.

[7] A. Weiss, J. Igelsboeck, S. Calinon, A. Billard, and M. Tscheligi, "Teaching a humanoid: A user study on learning by demonstration with hoap-3," in *IEEE Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2009, pp. 147–152.

[8] J. Huang and M. Cakmak, "Code3: A system for end-to-end programming of mobile manipulator robots for novices and experts," in *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*. ACM, 2017, pp. 453–462.

[9] M. Stenmark, M. Haage, and E. A. Topp, "Simplified programming of re-usable skills on a safe industrial robot: Prototype and evaluation," in *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*. ACM, 2017, pp. 463–472.

[10] D. F. Glas, T. Kanda, and H. Ishiguro, "Human-robot interaction design using Interaction Composer: Eight years of lessons learned," in *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*. IEEE Press, 2016, pp. 303–310.

[11] K. Fischer, F. Kirstein, L. C. Jensen, N. Krüger, K. Kukliński, T. R. Savarimuthu *et al.*, "A comparison of types of robot control for programming by demonstration," in *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*. IEEE Press, 2016, pp. 213–220.

[12] P. E. Rybski, K. Yoon, J. Stolarz, and M. M. Veloso, "Interactive robot task training through dialog and demonstration," in *Proceedings of the ACM/IEEE international conference on Human-robot interaction*. ACM, 2007, pp. 49–56.

[13] T. Lozano-Pérez, "Robot programming," *Proceedings of the IEEE*, vol. 71, no. 7, pp. 821–841, 1983.

[14] J. Aleotti, S. Caselli, and M. Reggiani, "Leveraging on a virtual environment for robot programming by demonstration," *Robotics and Autonomous Systems*, vol. 47, no. 2, pp. 153–161, 2004.

[15] T. Zhang, Z. McCarthy, O. Jow, D. Lee, K. Goldberg, and P. Abbeel, "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation," *arXiv preprint arXiv:1710.04615*, 2017.

[16] S. Calinon and A. Billard, "Learning of gestures by imitation in a humanoid robot," Cambridge University Press, Tech. Rep., 2007.

[17] M. Wächter, S. Schulz, T. Asfour, E. Aksoy, F. Wörgötter, and R. Dillmann, "Action sequence reproduction based on automatic segmentation and object-action complexes," in *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE, 2013, pp. 189–195.

[18] A. Billard and M. J. Matarić, "Learning human arm movements by imitation:: Evaluation of a biologically inspired connectionist architecture," *Robotics and Autonomous Systems*, vol. 37, no. 2-3, pp. 145–160, 2001.

[19] A. Ude, C. G. Atkeson, and M. Riley, "Programming full-body movements for humanoid robots by observation," *Robotics and autonomous systems*, vol. 47, no. 2-3, pp. 93–108, 2004.

[20] D. Martinez and D. Kragic, "Modeling and recognition of actions through motor primitives," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, 2008, pp. 1704–1709.

[21] B. C. Stadie, P. Abbeel, and I. Sutskever, "Third-person imitation learning," *arXiv preprint arXiv:1703.01703*, 2017.

[22] Y. Liu, A. Gupta, P. Abbeel, and S. Levine, "Imitation from observation: Learning to imitate behaviors from raw video via context translation," *arXiv preprint arXiv:1707.03374*, 2017.

[23] S. Calinon and A. Billard, "Statistical learning by imitation of competing constraints in joint space and task space," *Advanced Robotics*, vol. 23, no. 15, pp. 2059–2076, 2009.

[24] ——, "Stochastic gesture production and recognition model for a humanoid robot," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2769–2774.

[25] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, 2009, pp. 763–768.

[26] P. Sermanet, C. Lynch, J. Hsu, and S. Levine, "Time-contrastive networks: Self-supervised learning from multi-view observation," *arXiv preprint arXiv:1704.06888*, 2017.

[27] T. Yu, C. Finn, A. Xie, S. Dasari, T. Zhang, P. Abbeel, and S. Levine, "One-shot imitation from observing humans via domain-adaptive meta-learning," *Robotics: Science and Systems (RSS)*, 2018.

[28] S. Niekum, S. Osentoski, G. Konidaris, S. Chitta, B. Marthi, and A. G. Barto, "Learning grounded finite-state representations from unstructured demonstrations," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 131–157, 2015.

[29] A. Mohseni-Kabir, C. Rich, S. Chernova, C. L. Sidner, and D. Miller, "Interactive hierarchical task learning from a single demonstration," in *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*. ACM, 2015, pp. 205–212.

[30] K. Ramirez-Amaro, M. Beetz, and G. Cheng, "Transferring skills to humanoid robots by extracting semantic representations from observations of human activities," *Artificial Intelligence*, vol. 247, pp. 95–118, 2017.

[31] Z. Zeng, Z. Zhou, Z. Sui, and O. C. Jenkins, "Semantic robot programming for goal-directed manipulation in cluttered scenes," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7462–7469.

[32] M. Racca and V. Kyrki, "Active robot learning for temporal task models," in *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*. ACM, 2018, pp. 123–131.

[33] S. Alexandrova, M. Cakmak, K. Hsiao, and L. Takayama, "Robot programming by demonstration with interactive action visualizations," in *Robotics: Science and Systems (R:SS)*, 2014, pp. 48–56.

[34] W. Goo and S. Niekum, "Learning multi-step robotic tasks from observation," *arXiv preprint arXiv:1806.11244*, 2018.

[35] M. Wüthrich, P. Pastor, M. Kalakrishnan, J. Bohg, and S. Schaal, "Probabilistic object tracking using a range camera," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Nov. 2013, pp. 3195–3202.

[36] A. Walsman, W. Wan, T. Schmidt, and D. Fox, "Dynamic high resolution deformable articulated tracking," *3D Vision*, 2017.

[37] Y. Xiang and D. Fox, "DA-RNN: Semantic mapping with data associated recurrent neural networks," *arXiv preprint arXiv:1703.03098*, 2017.

[38] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

[39] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, pp. 239–256, 1992.