Representing and Learning Complex Object Interactions

Yilun Zhou and George Konidaris
Duke Robotics
Departments of Computer Science and Electrical & Computer Engineering
Duke University, Durham, North Carolina 27708
{yilun, gdk}@cs.duke.edu

***Abstract*—We present a framework for representing scenarios with complex object interactions, in which a robot cannot directly interact with the object it wishes to control, but must instead do so via intermediate objects. For example, a robot learning to drive a car can only indirectly change its pose, by rotating the steering wheel. We formalize such complex interactions as chains of Markov decision processes and show how they can be learned and used for control. We describe two systems in which a robot uses learning from demonstration to achieve indirect control: playing a computer game, and using a hot water dispenser to heat a cup of water.**

I. Introduction

Powerful task representations will be essential for constructing generally capable robots, because they will need to reason about how to interact with the world to achieve their goals. While there has been a great deal of research on reasoning about such interactions, it has largely focused on the case where a robot directly manipulates an object of interest; for example, the robot must open a door [16] or grasp and fold a towel [15].

However, in many cases the direct manipulation of the object that the robot is interacting with is not the point; rather, the robot should use that object to indirectly affect the state of some other object(s). Consider teaching a robot to drive a car. The robot's direct interaction with the environment is via the steering wheel. However, we are not trying to teach the robot to move the steering wheel to a specific position, or even to follow a specific steering-wheel trajectory; instead, we are *using the steering wheel to control the car*—controlling the *car* is the primary objective of our actions, but this must be achieved through interaction with an intermediate object (the wheel). We cannot learn only the interaction with the intermediate object (because that ignores the state of the car), but we cannot ignore it either (because it is the only way to control the car).

A representation that is aware of this interaction structure could facilitate learning in at least two ways. First, learning each part of the interaction can be done independently, in an appropriate state space: when the robot learns how its grippers rotate the steering wheel, it need not care about the car position; when it learns how the wheel controls the car, it does not even need to be in the driver's seat (observing a human driver is sufficient). Second, learned knowledge can be transferred even if parts of the task change: if the robot must now learn to steer a ship, it need only focus on the interaction between the steering wheel and the ship, without having to re-learn how to rotate the wheel. This transfer of knowledge could substantially reduce the time required to learn everyday tasks.

We therefore introduce a framework for representing such complex interactions as chains of Markov decision processes, and show how such a chain can be learned. We show that this framework allows a robot to learn to use a joystick to control a video game from demonstration. In addition, we extend the framework to handle more general object interaction graphs, which express interaction relationships that can change over time. For example, a robot may pour cold water from bottle to an electric kettle, boil the water, and pour the water back from the kettle to the bottle. In the first stage of the task, the robot uses the bottle to affect the state of the kettle (i.e. amount of water inside), but in the final stage, it uses the kettle to affect the state of the bottle. We introduce the use of *activation classifiers* and *deactivation classifiers* to represent the circumstances under which an interaction between two objects becomes active or inactive, and show that our framework can be used to re-sequence individual skills learned by demonstration to operate a hot water dispenser to warm a cup of cold water.[1]

II. Background and Related Work

Control learning problems are often modeled as Markov decision processes (or MDPs) [18], which can be described by a tuple $\{S, A, Tr, R\}$ where S is a set of (possibly continuous) states, A is a set of (possibly continuous) actions, $Tr : S \times A \to \Pr(S)$ is a transition function that maps current state and action to a distribution over next state, and $R : S \times A \to \mathbb{R}$ is a reward function that maps current state and action to a real-valued reward. A solution to an MDP takes the form of a policy $\pi : S \to A$ that maps a state to an action to be taken in that state, in order to maximize the *return* (expected discounted summed reward):

$$\arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right], \tag{1}$$

[1]A video explaining the model and showing both experiments is available at https://www.youtube.com/watch?v=TJlxXX1v6S4.

where $0 < \gamma \leq 1$ is a discount factor expressing a preference for immediate over delayed reward (set to 1 in this paper).

In learning from demonstration (LfD) [2], a robot is given demonstration trajectories obtained by executing an expert's policy, and must be able to reproduce the policy. There are multiple possible strategies here, depending on how much the robot knows about the MDP. For example, if the robot does not know the transition or reward function, it may try to learn the policy directly, using the state-action pairs in the demonstration trajectories as labelled training examples in a supervised learning setting. If the robot knows the transition model but neither knows nor observes the reward function, it could produce a policy that tries to follow the demonstrated trajectories, or use inverse reinforcement learning [1] to try to recover the reward function. If the robot is given the reward function, it may also use the state-action-reward tuples in the demonstration trajectories as labelled training data to learn the MDP transition and reward functions, and then solve the resulting MDP. LfD has received a great deal of attention [2, 4], but to the best of our knowledge we are the first to explicitly consider object-object interactions in the same manner as robot-object interactions.

In our interaction graph model, demonstrations are broken into different pieces (e.g. pressing a button, moving a cup, pouring water, etc.), and each segment can be considered a task with transition dynamics independent from state of other objects in the absence of collisions. Much work has been done on automatically breaking unstructured demonstrations into subskills [9, 14, 6, 5, 8, 3, 13, 11, 17], which could be applied in our framework to find the individual motor skills that enable or disable an interaction.

For scenarios with multiple objects, Ekvall and Kragic [7] identified spatio-temporal constraints for a pick-and-place task involving multiple objects from either teacher instruction or inference and reasoning over multiple demonstrations. Our activation and deactivation classifier have similar functions to the constraints in their work. In a similar vein, Konidaris et al. [12] used classifiers to represent the conditions under which a high-level action can be executed, and used them to construct an abstract representation of a task.

There is a large body of literature on building complex skills via hierarchical learning and execution. Kolter et al. [10] introduced hierarchical apprenticeship learning which enables the trainer to demonstrate skills at different levels of abstraction. For example, a trainer can demonstrate footstep placement on rough terrain, and also demonstrate how to do low-level motor control to locally follow the footsteps for quadruped locomotion.

In general, while hierarchical approaches are concerned with hierarchical structure that is *internal to the robot*, we are concerned with structure that exists in the relationships between objects in the world. In that sense, hierarchical approaches could be viewed as *vertical* (the robot building new skills on top of old skills), whereas our approach could be considered *horizontal* (the robot affecting control through chains of objects).

III. Interaction Chains

We now present a model that captures scenarios in which a robot interacts with an object through a chain of other objects. The state of each object in the chain affects the transition dynamics of its successor object, and is therefore modeled as an action in the successor object's MDP. Later, Section V generalizes our interaction chain model to a graph, which can capture scenarios with more complex interactions.

An interaction chain consists of N objects $O_1, O_2, ..., O_N$, where O_1 is the robot. Each O_i has an associated MDP $M_i \equiv \{S_i, A_i, Tr_i, R_i\}$, where S_i is the set of states for O_i, A_i is the set of "actions" (explained below), $Tr_i : S_i \times A_i \rightarrow \Pr(S_i)$ is a (stochastic) transition function, and $R_i : S_i \rightarrow \mathbb{R}$ is an optional reward (alternatively, cost) function.

In the chain model, we assume that interactions only exist between successive objects O_{i-1} and O_i, for all $i \in \{2, ..., N\}$. For such an interaction, we call O_{i-1} the *predecessor object* and O_i the *successor object*. The interactions are modeled by coupling their MDPs so that the state of object $i-1$ affects (or controls) the state of object i, *and thus serves as the action of* M_i, i.e.:

$$S_{i-1} \equiv A_i \;\; i = 2, ..., N. \tag{2}$$

S_1 describes the state of the robot, and A_1 describes the raw control available to it. Changes in the robot state (along with the passive dynamics at each level) are ultimately the source of all state transitions in our model, as objects interact with each other through the chain. We make no particular distinction between the robot and other objects, except that the robot is the root of the chain and cannot be controlled by other objects.

In the car-driving task, $S_1 \equiv A_2$ is the state of the robot (e.g. gripper position), $S_2 \equiv A_3$ is the state of the steering wheel (e.g. rotation angle), and S_3 is the state of the car (e.g. pose and speed). Figure 1 shows the interaction chain for this example.

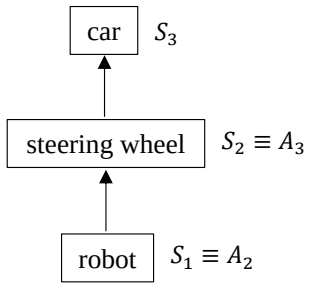


Fig. 1. The interaction chain for the car driving example.

During implementation we discovered a subtle point about timing: since most physical systems are continuous in nature but we are using discrete time, we found that when modeling the interaction between successive objects O_{i-1} and O_i, as a practical matter the action causing the transition from $s_i^{(t)}$ to $s_i^{(t+1)}$ is better predicted by $s_{i-1}^{(t+1)}$ than by $s_{i-1}^{(t)}$. Recall that

the state of the predecessor object serves as the action of the successor object. Thus for O_{i-1} and O_i, we have $S_{i-1} \equiv A_i$. Therefore, for two consecutive time points, t and $t+1$, we have four states: $s_{i-1}^{(t)}$, $s_i^{(t)}$, $s_{i-1}^{(t+1)}$, $s_i^{(t+1)}$. It may seem natural to credit $s_{i-1}^{(t)}$ for making the transition in O_i from $s_i^{(t)}$ to $s_i^{(t+1)}$. However, it is actually better to treat $s_{i-1}^{(t+1)}$ as the action for the transition. For example, consider the interaction between the robot and the steering wheel. When the robot holds the steering wheel, $s_{\text{robot}}^{(t+1)}$, the hand position at time $t+1$, instead of $s_{\text{robot}}^{(t)}$, best predicts $s_{\text{wheel}}^{(t+1)}$, the steering wheel rotation angle at time $t+1$. More generally, at time t, it is difficult to predict the action in effect for the duration between time t and $t+1$, which is revealed at time $t+1$.

Using this notation, Figure 2 shows a diagram of state transitions at all levels. Nodes that do not have arrows represent initial conditions or robot actions; all others have two incoming arrows representing the two inputs to the transition function. The top row denotes the robot actions.

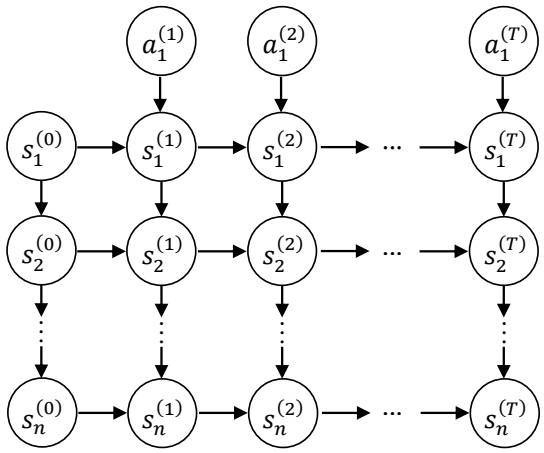


Fig. 2. The forward transition graph for an interaction chain.

During learning and control, a key piece of information is the *inverse transition function*, $Tr^{-1} : S_i \times S_i \rightarrow S_{i-1}$. Given a state transition of the successor object, the inverse transition function outputs a state in the predecessor object that can induce the transition. This function can be either derived from the forward transition function Tr, if known, or learned. Note that the inverse transition function presents some difficulties. There may be several possible predecessor states that induce the target successor transition, and it may return a state that is not reachable from the predecessor object's current state. Moreover, in a stochastic system, it may only return a state the is *likely* (rather than guaranteed) to cause the target transition. Fortunately, a relatively simple recursive feedback control algorithm (given in the Appendix) was sufficient to overcome these difficulties in our experiments.

IV. The Car Game

The car game domain involves a robot operating a joystick that controls a car in a custom-made video game. The interaction chain diagram was shown previously in Figure 1, with the steering wheel being replaced by the joystick. We use a mix of autonomous learning and learning from demonstration to show that the robot can learn to control the car to follow complex trajectories.

Because of the small movements and high sensitivity of the joystick, we do not use computer vision. Instead, the robot hand position is extracted directly from the robot API; the joystick configuration, also retrieved programmatically, is represented by two axis (left/right and forward/backward) values in the range of -1 to 1. The position of the car is also read programmatically.

We learn two interactions: between the robot hand and the joystick, and between the joystick and the car. We collect the data for the first interaction (robot to joystick) by having the robot directly play with the joystick and learn the association between the hand position and the joystick axis values. The robot tried about 500 hand positions to learn the relationship between the hand position and the joystick configuration. This interaction is not linear as the joystick axis values do not change linearly with tilting angle.

We used human demonstration for the second interaction (joystick to car). While the game runs at 30Hz, we found that the robot can only be controlled reliably at 3Hz: beyond this control rate, the robot will start dropping control commands and cannot faithfully execute a command sequence. Thus, the training data are also collected at 3Hz. We generated about 500 transition pairs. The game dynamics were designed to be noisy and non-linear.

For both levels, we directly learned the inverse transition function using polynomial regression. Figure 3 shows the learning process.

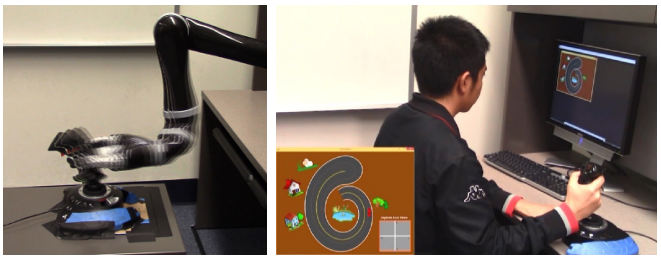


Fig. 3. Learning at two levels. Left: learning the robot-joystick interaction by playing with the joystick. Right: learning the joystick-car interaction by observing a human playing the game. (Best viewed in color.)

During execution, the robot's goal is to follow a given car path as closely as possible. Three complex trajectories were followed by the robot. Figure 4 shows the followed trajectories overlaid on the game screen in the left column, and the commanded and executed paths in the right column.

We calculated the average distance between demonstrated and followed trajectories by sampling random points from the

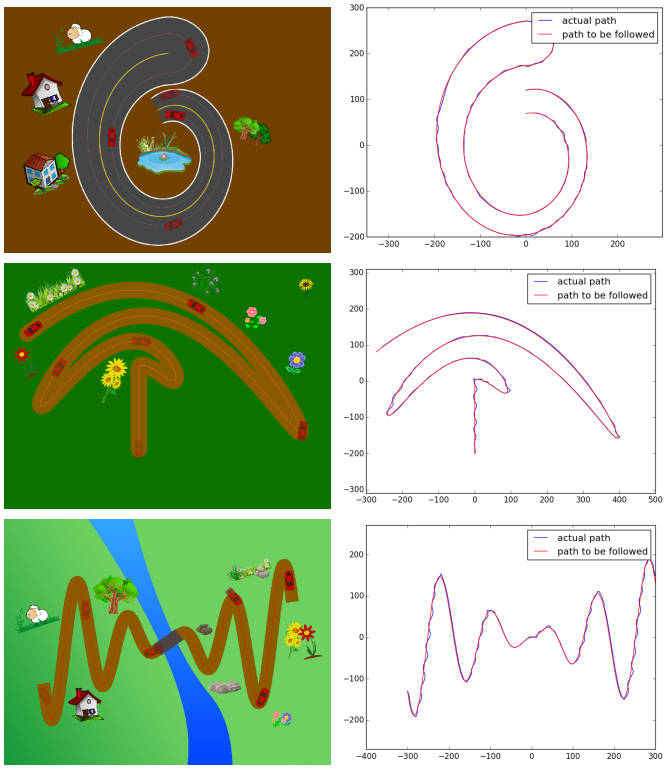


Fig. 4. Trajectory following. The car images are faded out with time so that positions older in time are more transparent. The "road" on the game interface is for visual illustration purposes only and does not constrain the motion of the car. (Best viewed in color.)

executed trajectory, finding the minimum distance of each point to the commanded trajectory (also discretized), and averaging those distances. The paths have average error of 0.790, 1.2480, and 1.8662 pixels (on a 800×600 game screen), respectively. Thus, we are able to reproduce the demonstrated trajectories very accurately on this task.

Our model is readily adaptive to several variations: if the joystick position is changed, the robot need only translate its hand accordingly; if the robot gripper is changed, it need only re-learn how to hold and control the joystick; if the game is changed, the robot need only re-learn the interaction between the joystick and the new game.

V. Interaction Graphs

The model presented in Section III can only describe scenarios in which objects are arranged in a chain. However, many real-life scenarios are more complex. For example, consider operating a microwave oven (consisting of a door, an on/off button, and a power level knob) to heat food. The door must be open to place and retrieve food, and closed for the oven to operate. The on/off button starts and stops the oven, and the power level knob changes the heating speed. All three components can influence the temperature of the food (the door influences the temperature by preventing it from changing when open). However, they do not interact with each other. Moreover, in addition to interacting with the components of the microwave, the robot can directly modify properties of the food such as its position and orientation. We must therefore extend the interaction chain to a graph where each node represents an object (a collection of state variables that can be simultaneously manipulated). A directed edge from node i to node j represents the fact that the state of node i (possibly jointly with states of other nodes) can affect the state of node j. Figure 5 shows the interaction graph for the microwave oven example described above.

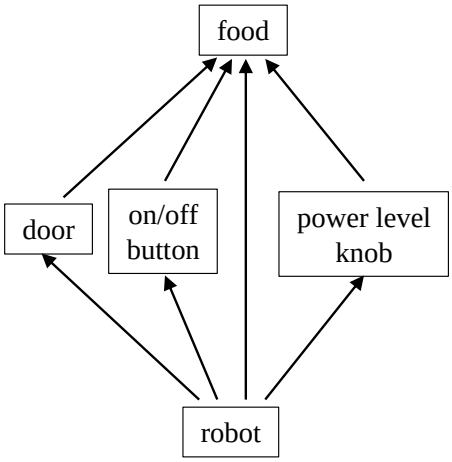


Fig. 5. The interaction graph for the microwave oven example.

We represent an interaction graph as a collection of MDPs $M = \{M_1, ..., M_n\}$, along with a graph $G(M, E)$ in which the directed edges between the MDPs represent relationships of influence. Since many objects can jointly affect one object, the action set for object i is now the Cartesian product of all the state sets belonging to its parents:

$$A_i \equiv \prod_{k:(k,i)\in E} S_k. \tag{3}$$

When multiple objects can jointly interact with one object, the inverse transition function can be even harder to compute. To mitigate the problem, we distinguish between active and inactive interactions. In an interaction chain, we assume that all interactions are always "happening". For example, as long as the robot is holding the steering wheel, any movement of the hand will induce corresponding rotation of the wheel. However, in an interaction graph, an object may only affect another in some cases. For example, the robot can only manipulate one object at a time. The states of objects that are not being manipulated progress according to their passive dynamics.

Activation classifiers and *deactivation classifiers* govern the change in interactions from inactive to active, and from active to inactive, respectively. An activation classifier returning `true` denotes the activation of a previously inactive interaction, and `false` means that the interaction remains inactive. The deactivation classifier is similar.

As a specific example, in the microwave oven scenario, the interaction between the robot and the food is only active when the robot is grasping the food. The interaction between the power level knob and the food (i.e. changing the heating speed) is only active if the food is inside the microwave oven, the door is closed and the on/off button is on (so that heating process can occur).

In learning, both the transition function and (de)activation classifiers must be learned. The transition function should only be learned when the interaction is active. To learn the interaction classifiers, the robot needs the interaction status between two objects at consecutive time steps. It is given by hand in our example, but could in principle be learned autonomously by determining whether or not the passive dynamics of the object have been perturbed.

VI. The Water Dispenser

We now demonstrate the use of an interaction graph in a system where a robot uses a hot water dispenser to turn a cup of cold water into a cup of hot water. The water dispenser has a water tank to hold water, a dispense button to dispense the water in the tank, and a power button to heat the water in the tank. Figure 6 shows the interaction graph.

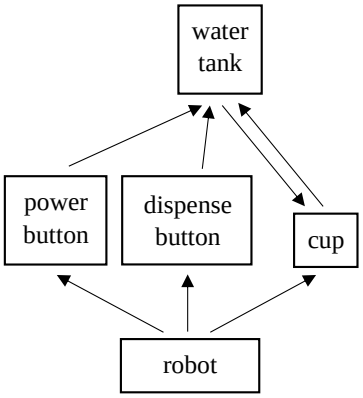


Fig. 6. The interaction graph of the water dispenser experiment.

We used an Intel Realsense F200 camera to track the cup pose and the depressions of power and dispense buttons. Since it is very challenging to extract information about liquid (such as the water amount in a tilted cup) and computer vision is not a focus of our work, we pre-computed the water amount for each cup tilting angle. We also estimated the water temperature in the tank during heating.

For the robot hand, the state variables include x, y, z coordinates in the robot frame, and the tilt angle θ necessary for pouring water from cup. We use a binary variable to model whether the fingers of the hand are open or closed. The state variables of the cup include its pose variables (defined similarly to those for the hand, but in the camera frame), and two real variables between 0 and 1 denoting the normalized amount of water (0 being empty) and the water temperature (0 being cold). Finally, it has a change in water level defined to be $\Delta\text{water\_amount}^{(t)} \equiv \text{water\_amount}^{(t)} - \text{water\_amount}^{(t-1)}$. Only with the inclusion of this state variable can the interaction between cup and water tank be Markov. The two buttons each have a level of depression expressed as an integer between 0 and 6, since they have a maximum depression of 6 pixels in our camera setting. The water tank has three variables: water amount, water temperature, and change in water amount, defined similarly as those of the cup.

For the robot-cup interaction, activation occurs when the cup is inside the hand and the hand is closed; deactivation occurs when the hand opens. When the interaction is active, the position and tilt of the cup will follow those of hand. They are not the same, however, since they are in different reference frames. The water amount will decrease and the change in water amount will be negative if the cup is tilted. The water temperature does not change in this interaction.

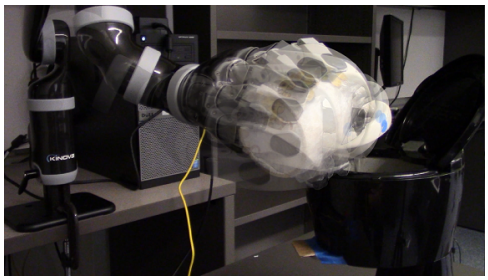
For the interactions between the robot and the power/dispense button, activation occurs when the robot hand is on top of the button and deactivation happens at the same position. During interaction, the button level will be determined by the z-coordinate of the hand (its height). For both buttons, a depression greater than 3 will trigger heating and dispensing, respectively. When the dispense button is active, the water amount in the tank decreases by $1/16$ each time unit (second) until it reaches 0, since it takes about 16 seconds to empty the dispenser. When the power button is active, the temperature increases by $1/80$ until reaching 1, for a similar reason.

Both pouring and dispensing interactions can happen between the cup and tank, albeit in different directions. The pouring interaction activates when the cup is above the tank and the dispensing interaction activates when cup is below the tank. The key to the transition function is that the water amounts *change* in opposite directions. Thus, the Markov property holds through the change in water amount. Finally, the new water temperature is an average between the old temperature and the temperature of the newly added water, weighted by their relative volumes.

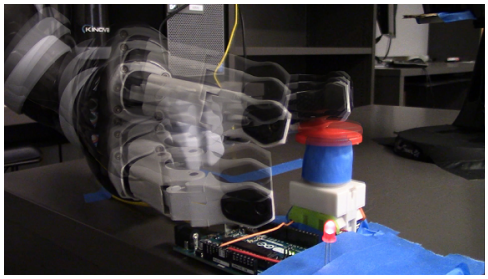
During learning, the robot was tele-operated to do a set of tasks that helped it learn all the interactions. Figure 7 visualizes the tele-operation. About ten demonstration pieces were given. It should be noted that these skill demonstrations are *unordered*. The active and inactive states are hand-labeled. From these skills, the robot learns a model of the interaction between the objects.

Given this model, the robot was able to apply a control algorithm to obtain a cup of hot water when it is given a cup of cold water, a challenging task that requires multiple object interaction steps. Specifically, in execution, the robot must use the cup to pour the cold water into the water tank, move the cup to below the tank, press and hold the power button until the water is boiling, then press the dispense button. Once the water tank is empty, the robot must retrieve the cup of newly heated water.

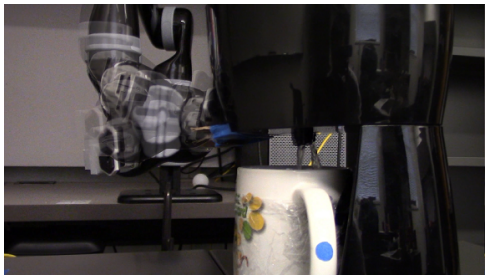
After we modified the control algorithm for the interaction chain model to incorporate activation and deactivation classi-



(a) Pouring water

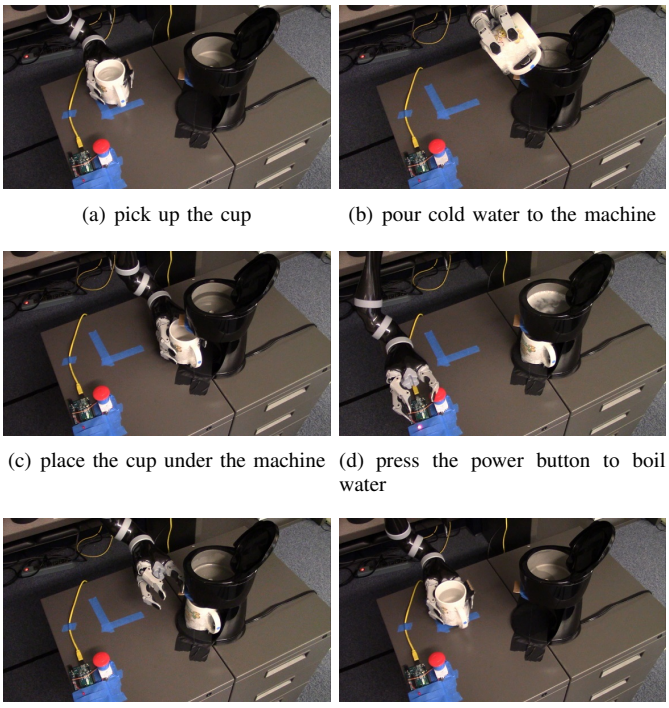


(b) Pressing the power button



(c) Pressing the dispense button

Fig. 7. The robot learns various interactions through tele-operation. (Best viewed in color.)



(a) pick up the cup (b) pour cold water to the machine (c) place the cup under the machine (d) press the power button to boil water (e) press the dispense button to dispense water (f) move the cup to original position

Fig. 8. The robot autonomously sequences learned skills to heat a cup of water. (Best viewed in color.)

fiers (see the Appendix), our system was able to autonomously achieve the goal using its learned knowledge. The assembled execution consists of about 1000 discrete control steps. Figure 8 shows several waypoints in the robot execution.

VII. Conclusion and Future Work

We have introduced a flexible representation that goes beyond modeling robot-object interactions to account for object-object interactions, and showed that it can be used to learn two distinct real-life tasks involving complex object interactions.

There are several advantages of our model. First, it naturally represents interactions among objects, so that we can build complex systems capable of using intermediate objects. In particular, tool use is an embodiment of this characteristic as tools are intermediate objects we use to achieve some goal. In order to use such systems, robots must also be able to reason about interactions between objects.

Second, by modeling each object as a separate MDP, we can factor the joint state space to mitigate against the curse of dimensionality. Models that are unaware of intermediate objects must represent the compounded interaction of the robot monolithically. Even if individual interactions are simple, the compounded one can be very hard to model.

Finally, our model can accomplish knowledge transfer. Since each interaction is represented by an MDP, a control algorithm can directly use a transferred MDP in place of a freshly learned MDP for task execution. This step requires no overhead and directly reduces the total amount of learning, which can be very significant for low-cost robots with limited onboard computational resources. Since our work does not require demonstrations to be given in the correct order, interactions can be learned at different times and from different sources. In addition, when equipped with a database of "primitive interaction models" as prior knowledge, a robot can be immediately versatile in everyday environments, without too much learning.

Other than a better control algorithm, autonomous interaction graph structure discovery is a very important direction for future work. The inclusion of prior knowledge (e.g. our interactions are causal and it is highly unlikely that a lamp lighting up will cause a button to depress), accurate sensing, and active information gathering will likely prove necessary for learning complex interaction model structures completely from scratch. In addition, ideas from structure discovery in Bayesian networks may also be relevant here.

ACKNOWLEDGEMENT

We thank the anonymous reviewers for their helpful suggestions and criticisms. We are grateful for Ying Qi for narrating the video. The game sprites and pictures appearing in the video are courtesy of www.clipartbest.com, www.clipartsheep.com, www.clker.com, www.flickr.com, www.iconarchive.com, www.openclipart.org, www.pd4pic.com, and www.pixabay.com under Creative Commons 0, Creative Commons Attribution License, or custom license allowing free non-commercial use with attribution. This research was supported in part by DARPA under agreement number D15AP00104, and by the National Institutes of Health under award number R01MH109177. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health or DARPA.

APPENDIX

In this section, we sketch the feedback-control algorithms used in the two experiments. They are very simple and are included only to make our work easier to reproduce. The development of more complex algorithms is left to future work.

A. Trajectory Following for Interaction Chains

The building block of the algorithm is a controller that moves the n-th object toward a target state; it can be used as the basis of a *feedback controller* that, over a sequence of successive calls, controls the system to approximately follow a desired trajectory.

We aim to learn an inverse transition function $Tr^{-1} : S_i \times S_i \to S_{i-1}$ such that for the interaction between object $i-1$ and object i, the function can return an action $s_{i-1}^{(t+1)}$ that can cause the desired transition from $s_i^{(t)}$ to $s_i^{(t+1)}$ (recall the time convention discussed at the end of Section III). Note that the returned $s_{i-1}^{(t+1)}$ can be impossible to reach from the current state of object $i-1$, $s_{i-1}^{(t)}$. For example, if the robot can only turn the steering wheel gradually, but $s_{i-1}^{(t)}$ and $s_{i-1}^{(t+1)}$ represents two very different angles, then we must effect the transition over a longer period of time. We use path planning at each level in turn to solve this problem. The resulting algorithm `move_to`, which should be called successively to achieve trajectory following, is given in Algorithm 1.

def `move_to`$(i,\ s_i)$
  while *not done* **do**
    `move_one_step` $(i,\ s_i)$;
  end
def `move_one_step`$(i,\ s_i)$
  if $i == 1$ **then**
    control robot toward s_i for a time unit;
  else
    $s_i^{(\text{next})}$ = `state_plan`$(i,\ s_i^{(\text{cur})},\ s_i)$;
    $s_{i-1}^{(\text{next})}$ = `inv_transition`$(i,\ s_i^{(\text{cur})},\ s_i^{(\text{next})})$;
    `move_to`$(i-1,\ s_{i-1}^{(\text{next})})$;
  end
def `inv_transition`$(i,\ s_i^{(t)},\ s_i^{(t+1)})$
  return $s_{i-1}^{(t+1)}$ *most likely for transition* $s_i^{(t)} \to s_i^{(t+1)}$;
def `state_plan`$(i,\ s_i^{(\text{cur})},\ s_i)$
  return *next state for* O_i *on the way to* s_i *from* $s_i^{(\text{cur})}$;

Algorithm 1: Feedback Control Algorithm. Superscript of "(cur)" means current value of the state.

B. Trajectory Following for Interaction Graphs

Trajectory following for a interaction graph is similar to that in the chain case, except that the robot must know how to activate and deactivate interactions. We model actions that perform the activation and deactivation—reaching states within the classifiers—as motor skills that can either be given by the system designer, or learned from demonstration. Specifically, from the given or learned classifiers the robot can find states of both manipulating and manipulated objects that will (de)activate an interaction. Then objects are manipulated such that the states will match the predicted states.

With (de)activation skills, the inverse transition function should also output the interaction type. For example, the transition of water amount increasing in the cup has the interaction type of dispensing, while the transition that the cup position changes has the interaction type of robot moving the cup. The only addition to the trajectory following algorithm is that when the interaction type switches (e.g. from hand moving cup to pressing a button), the previous interaction must be deactivated and the next interaction activated before the control algorithm loops.

REFERENCES

[1] P. Abbeel and A.Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*, 2004.

[2] B.R. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, May 2009.

[3] J. Butterfield, S. Osentoski, G. Jay, and O.C. Jenkins. Learning from demonstration using a multi-valued function regressor for time-series data. In *Proceedings of the Tenth IEEE-RAS International Conference on Humanoid Robots*, 2010.

[4] S. Chernova and A. Thomaz. *Robot Learning from Human Teachers*. Morgan & Claypool, 2014.

[5] S. Chiappa and J. Peters. Movement extraction by detecting dynamics switches and repetitions. In *Advances in Neural Information Processing Systems 23*, pages 388–396, 2010.

[6] S. Chiappa, J. Kober, and J. Peters. Using Bayesian dynamical systems for motion template libraries. In

Advances in Neural Information Processing Systems 21, pages 297–304, 2009.

[7] S. Ekvall and D. Kragic. Robot learning from demonstration: a task-level planning approach. *International Journal of Advanced Robotic Systems*, 5(3):223–234, 2008.

[8] D.H. Grollman and O.C. Jenkins. Incremental learning of subtasks from unsegmented demonstration. In *International Conference on Intelligent Robots and Systems*, 2010.

[9] O.C. Jenkins and M. Matarić. Performance-derived behavior vocabularies: data-driven acquisition of skills from motion. *International Journal of Humanoid Robotics*, 1 (2):237–288, 2004.

[10] J.Z. Kolter, P. Abbeel, and A.Y. Ng. Hierarchical apprenticeship learning with application to quadruped locomotion. In *Advances in Neural Information Processing Systems 20*, pages 769–776, 2008.

[11] G.D. Konidaris, S.R. Kuindersma, R.A. Grupen, and A.G. Barto. Robot learning from demonstration by constructing skill trees. *International Journal of Robotics Research*, 31(3):360–375, March 2012.

[12] G.D. Konidaris, L.P. Kaelbling, and T. Lozano-Perez. Constructing symbolic representations for high-level planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1932–1940, 2014.

[13] V. Krüger, D.L. Herzog, S. Baby, A. Ude, and D. Kragic. Learning actions from observations. *IEEE Robotics and Automation Magazine*, 17(2):30–43, 2010.

[14] D. Kulić, W. Takano, and Y. Nakamura. Online segmentation and clustering from continuous observation of whole body motions. *IEEE Transactions on Robotics*, 25 (5):1158–1166, 2009.

[15] J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *Proceedings of the 2010 IEEE Conference on Robotics and Automation*, pages 2308–2315, 2010.

[16] W. Meeussen, M. Wise, S. Glaser, S. Chitta, C. McGann, P. Mihelich, E. Marder-Eppstein, M. Muja, V. Eruhimov, T. Foote, J. Hsu, R.B. Rusu, B. Marthi, G. Bradski, K. Konolige, B. Gerkey, and E. Berger. Autonomous door opening and plugging in with a personal robot. In *Proceedings of the 2010 IEEE Conference on Robotics and Automation*, pages 729–736, 2010.

[17] S. Niekum, S. Osentoski, G.D.Konidaris, S. Chitta, B. Marthi, and A.G. Barto. Learning grounded finite-state representations from unstructured demonstrations. *The International Journal of Robotics Research*, 34(2): 131–157, 2015.

[18] R.S. Sutton and A.G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.