CST: Constructing Skill Trees by Demonstration

George Konidaris GDK@CSAIL.MIT.EDU
MIT CSAIL, 32 Vassar Street, Cambridge MA 02139 USA

Scott Kuindersma SCOTTK@CS.UMASS.EDU
Roderic Grupen GRUPEN@CS.UMASS.EDU
Andrew Barto BARTO@CS.UMASS.EDU
Computer Science Department, University of Massachusetts Amherst, Amherst MA 01003 USA

Abstract

We describe recent work on CST, an online algorithm for constructing skill trees from demonstration trajectories. CST segments a demonstration trajectory into a chain of component skills, where each skill has a goal and is assigned a suitable abstraction from an abstraction library. These properties permit skills to be improved efficiently using a policy learning algorithm. Chains from multiple demonstration trajectories are merged into a skill tree. We describe applications of CST to acquiring skills from human demonstration in a dynamic continuous domain and from both expert demonstration and learned control sequences on a mobile manipulator.

1. Introduction

Learning from demonstration (or LfD) (Argall et al., 2009) offers a natural and intuitive approach to robot programming: rather than investing effort into writing a detailed control program, we simply *show* the robot how to achieve a task. LfD has received a great deal of attention in recent years because it aims to facilitate ubiquitous general-purpose automation by removing the need for engineering expertise and instead enabling the direct use of existing human procedural knowledge.

This paper summarizes recent work on CST, an LfD algorithm with four properties which, taken together, distinguish it from previous work. First, rather than converting a demonstration trajectory into a single controller, CST segments demonstration trajectories into a sequence of controllers (which we term *skills*, but are also called behaviors or motion primitives). This aims to extract reusable components of the demonstrator's behavior. Second, CST extracts skills which have *goals*—in particular, the objective of skill n is to reach a configuration where skill $n+1$ can be successfully executed. Such skills can be refined by the robot using policy improvement algorithms. Third, CST optionally supports *skill-specific abstraction selection*, where each skill policy is defined using only a small number of relevant state and motor variables. This affords efficient representation and learning, facilitates transfer, and enables the acquisition of policies that are high-dimensional when represented monolithically but consist of subpolicies that can be individually represented using far fewer state variables. Finally, CST merges skill chains from multiple demonstrations into a *skill tree*, allowing it to deal with collections of trajectories that use different component skills to achieve the same goal, while also determining which trajectory segments are instances of the same policy.

2. Background

This work adopts the options framework—a hierarchical reinforcement learning formalism for learning and planning using temporally extended actions or *options*—for modeling acquired skills.

An option, o, consists of three components: an *option policy*, π_o, giving the probability of executing each action in each state in which the option is defined; an *initiation set* indicator function, I_o, which is 1 for states where the option can be executed and 0 elsewhere; and a *termination condition*, β_o, giving the probability of option execution terminating in states where the option is defined. Given an *option reward function* (often just a cost function with a termination reward), determining the option's policy can be viewed as just another reinforcement learning problem, and an appro-

Appearing in *Proceedings of the ICML Workshop on New Developments in Imitation Learning*, Bellevue, WA, USA, 2011. Copyright 2011 by the author(s)/owner(s).

priate policy learning algorithm can be applied. Once acquired, a new option can be added to an agent's action repertoire alongside its primitive actions, and the agent chooses when to execute it in the same way.

An option is a useful model for a robot controller: it contains all the information required to determine when a controller can be run (its initiation set), when it is done (its termination condition), and how it performs control (its policy). An option reward function allows us to model controllers that can be improved through experience. In the remainder of this paper, we will use the terms *skill* and *option* interchangeably.

CST performs segmentation based on a linear value function approximation. Each option value function, V, is thus approximated by a weighted sum of a given set of basis functions, $\phi_1, ..., \phi_n$: $\hat{V}(\mathbf{x}) = \sum_{i=1}^{n} w_i \phi_i(\mathbf{x})$. We use the Fourier basis (Konidaris et al., 2011b) throughout this work.

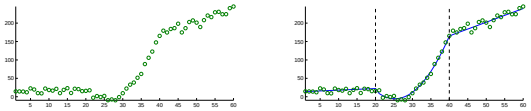
Although value function methods are widely used in reinforcement learning, robotics applications typically use *policy gradient algorithms*, which represent the policy π directly. Nevertheless, value function approximation is a key step in many policy gradient algorithms since an approximate value function can be used to obtain a low-variance estimator of the policy gradient when π is differentiable (Sutton et al., 2000). Even when it is not, an approximate value function is still a useful guide to the structure of the policy and often contains richer information for use in segmentation than the policy itself (which is often piecewise constant).

We may wish to define a skill policy in a smaller and more task-relevant state space than the full state space of the robot. This is known as an *abstraction*. In this work we define an abstraction M to be a pair of functions (σ_M, τ_M), where $\sigma_M : S \rightarrow S_M$ is a *state abstraction* mapping the overall state space S to a smaller state space S_M (often simply a subset of the variables in S, but potentially a more complex mapping involving significant feature processing), and $\tau_M : A \rightarrow A_M$ is a *motor abstraction* mapping the full action space A to a smaller action space A_M (often simply a subset of A). When using an abstraction, the agent's sensor input is filtered through σ_M and its policy π maps from S_M to A_M. We assume that each abstraction has a set of basis functions, Φ_M, defined over S_M which we can use to define a value function. Therefore, using an abstraction amounts to representing the relevant value function using that abstraction's basis functions.

3. CST

Given a demonstration trajectory, our task is to break it into component skills. A common principle in robotics, control and reinforcement learning is that the goal of a skill is to reach another skill (Lozano-Perez et al., 1984; Burridge et al., 1999; Tedrake, 2009; Konidaris and Barto, 2009). Thus, we aim to slice the trajectory into a chain of contiguous *skill segments*; we would like to split a segment into two when its value function is too complex to represent as a single segment, or when it is composed of two segments best represented with different abstractions.

Statistical *changepoint detection* algorithms perform just such a segmentation task. Here, we are given observed data and a set of candidate models. We assume that the data are sequentially generated by an instance of a single model, occasionally switching between models at certain points in time, called *changepoints*. We are to infer the number and positions of the changepoints and select and fit an appropriate model for each segment. Figure 1 shows a simple example.



(a) (b)

Figure 1. Artificial example data with multiple segments. The observed data (a) are generated by three different models plus noise (b; solid lines, changepoints shown using dashed lines). The first and third segments are generated by a linear model, whereas the second is quadratic.

Because our data are received sequentially and possibly at a high rate, we would like to perform changepoint detection online—processing transitions as they occur and then discarding them. Fearnhead and Liu (2007) introduced online algorithms for both Bayesian and maximum a posteriori (MAP) changepoint detection. We use the simpler MAP method.

Their model is as follows. A set, Q, of models is given with prior $p(q)$ for each $q \in Q$. Data tuples $(\mathbf{x}_t, y_t)$ are observed for times $t \in \{1, 2, \ldots, T\}$. The marginal probability of a segment length l is modeled with probability mass function $g(l)$ and cumulative distribution function $G(l) = \sum_{i=1}^{l} g(i)$. Finally, a segment from time $j+1$ to t can be fit using model q to obtain $P(j, t, q)$, the probability of the data segment conditioned on q. This results in a Hidden Markov Model where the hidden state at time t is the model q_t and the observed data is y_t given $\mathbf{x}_t$. This model is depicted in Figure 2. Notice that all of its transition

probabilities are known or computed directly from the data. Rather than attempting to learn the transition probabilities of the hidden states, we are instead trying to compute the maximum likelihood sequence of hidden states given their transition probabilities and the data. We can therefore use an online Viterbi algorithm to compute the most likely transition path through the hidden states in this HMM given the data, and thereby our segmentation.

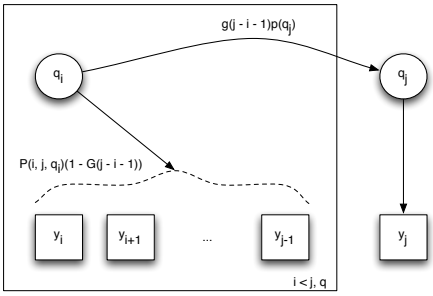


Figure 2. The Hidden Markov Model for changepoint detection. The model q_t at each time t is hidden, but produces observable data y_t. Transitions occur when the model changes, either to a new model or the same model with different parameters. The transition from model q_i to q_j occurs with probability $g(j-i-1)p(q_j)$, while the emission probability for observed data $y_i, ..., y_{j-1}$ is $P(i, j, q_i)(1 - G(j-i-1))$. These probabilities are considered for all times $i < j$ and models $q_i, q_j \in Q$.

Unfortunately, the number of possible paths in a Viterbi algorithm grows over time. However, most changepoint probabilities will be close to zero. We can therefore employ a particle filter to discard most of them and retain a constant number per timestep. We use the Stratified Optimal Resampling algorithm of Fearnhead and Liu (2007) for this purpose.

CST extracts skills by applying changepoint detection to the demonstrated trajectory data. It uses the sets of basis functions associated with each abstraction as models and the sample return, $R_t = \sum_{i=t}^{n} \gamma^{i-t} r_i$, from the state at each time t as the target variable. This effectively performs changepoint detection *on the value function sample obtained from the trajectory.* Segmentation thus breaks that value function sample up into simpler segments or detects a change in abstraction. This is depicted in Figure 3.

We assume a geometric distribution for skill lengths with parameter p, which gives us a natural way to set p via $k = 1/p$, the expected skill length. Since we are using linear value function approximation, we use a linear regression model with Gaussian noise as our model of the data. Following Fearnhead and Liu (2007), we assume conjugate priors: the Gaussian noise prior has

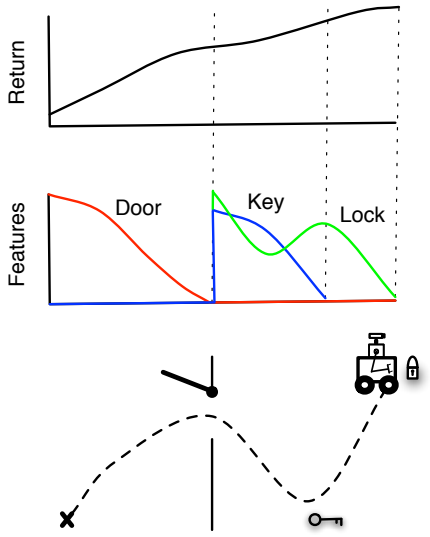


Figure 3. An illustration of a trajectory segmented into skills by CST. A robot executes a trajectory where it goes through a door, approaches and picks up a key, and then takes the key to a lock (bottom). The robot is equipped with three possible abstractions: state variables giving its distance to the doorway, the key, and the lock, respectively. The values of these variables change during trajectory execution (middle) as the distance to each object changes while it is in the robot's field of view. The robot also obtains a sample of return for each point along the trajectory (top). CST splits the trajectory into segments by finding a MAP segmentation such that the return estimate is best represented by a piecewise linear value function where each segment is defined over a single abstraction. Changepoints are indicated by dashed vertical lines.

mean zero and an inverse gamma variance prior. Note that we are using each R_t as the target regression variable in this formulation, even though we only observe r_t for each state. However, the relevant sufficient statistics can be computed incrementally using r_t, and thus the fit probability can be computed online at each timestep without storing any transition data.

Given multiple skill chains obtained this way from different trajectories, we would like to merge them into a skill tree by determining which pairs of trajectory segments belong to the same skills and which are distinct.

Because we wish to build skills that can be sequentially executed, we only consider merging two segments when they have the same target—which means that their goals are either to reach the initiation set of the same target skill, or to reach the same final goal. This means that we can consider merging the final segment of each trajectory, or two segments whose successor segments have been merged. Thus, two chains are merged by starting at their final skill segments. Each

pair of segments are merged if they are a good statistical match. This process is repeated until a pair of skill segments fail to merge, after which the remaining skill chains branch off on their own. This process is depicted in Figure 4. A similar process can be used to merge a chain into an existing tree by following the chain with the highest merge likelihood when a branch in the tree is reached. For more details, see Konidaris et al. (2010).

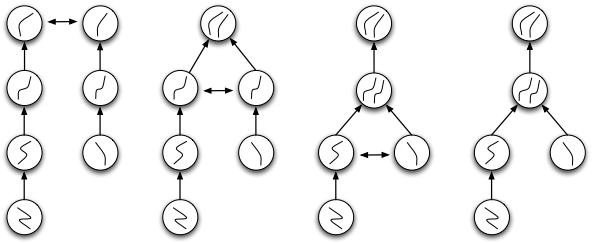


Figure 4. Merging two skill chains into a skill tree. First, the the final trajectory segment in each of the chains is considered (a). If these segments use the same model, overlap, and can be well represented using the same function approximator, they are merged and the second segment in each chain can be considered (b). This process continues until it encounters a pair of segments that should not be merged (c). Merging then halts and the remaining skill chains form separate branches of the tree (d).

4. Applications

4.1. The Pinball Domain

The Pinball domain is a difficult continuous domain with dynamic aspects, sharp discontinuities, and extended control characteristics.[1] The goal is to maneuver a small ball (which starts in one of two places) into a large red hole. The ball is dynamic, so its state is described by four variables: x, y, $\dot{x}$ and $\dot{y}$. Collisions with obstacles are fully elastic and cause the ball to bounce, so rather than merely avoiding obstacles the agent may choose to use them to efficiently reach the hole. There are five primitive actions: incrementing or decrementing $\dot{x}$ or $\dot{y}$ by a small amount (which incurs a reward of -5 per action), or leaving them unchanged (which incurs a reward of -1 per action). Reaching the goal obtains a reward of $10,000$.

Five pairs of demonstration trajectories (one trajectory in each pair for each start state) were collected from a human expert. Trajectory segmentation was successful for all demonstration trajectories, and all pairs were merged successfully into skill trees. Example segmentations are shown in Figure 5, and the resulting initiation sets (learned using logistic regression, where states inside the skill segment are positive examples and all other observed states are negative examples) are shown in Figure 6.

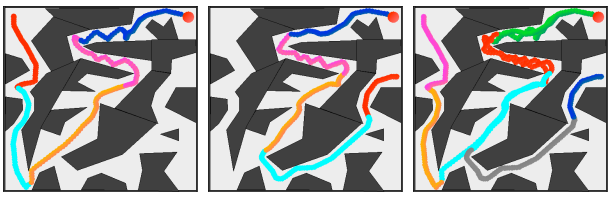


Figure 5. Demonstration trajectories segmented into skill chains, and the trajectory assignments obtained when the two chains are merged.

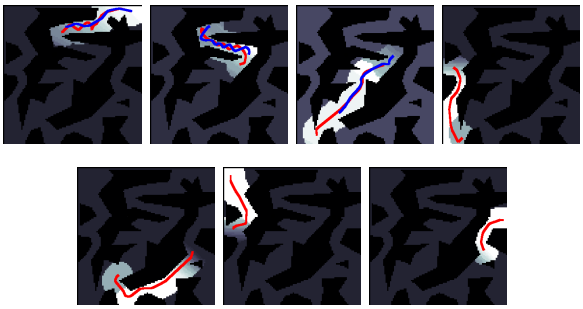


Figure 6. The initiation sets for each option in the tree shown in Figure 5.

Figure 7 shows learning curves comparing agents given skill trees extracted by CST, agents that build skill trees from scratch using skill chaining (Konidaris and Barto, 2009), and agents given skills pre-learned using 250 episodes of skill chaining. The agents given CST skill trees are able to learn very good policies within 10 episodes, by which time they even exceed the performance of agents given pre-learned skills. This is likely because agents with pre-learned skills begin with many skills to learn how to sequence whereas the CST agents identify only the number required to solve the task. For more details see Konidaris et al. (2010).

4.2. Acquiring Mobile Manipulation Skills from Human Demonstration

We next show that CST can scale up by applying it to acquire skill chains from human demonstration data on the uBot-5, a dynamically balancing mobile manipulator. The robot's task in this section is to enter a corridor, approach a door, push the door open, turn right into a new corridor, and finally approach and push on a panel (illustrated in Figure 8). A human operator provided 12 demonstration trajectories.

[1] Java source code for Pinball can be downloaded at: http://www-all.cs.umass.edu/~gdk/pinball

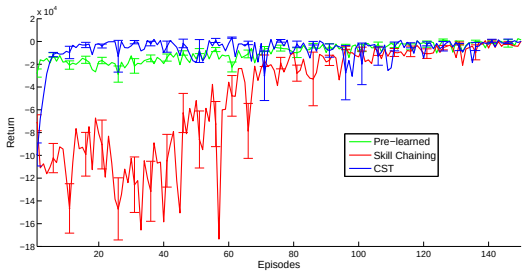
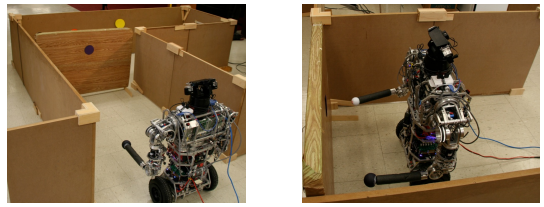
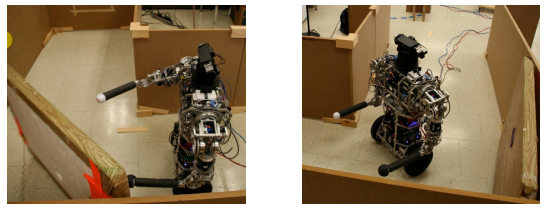


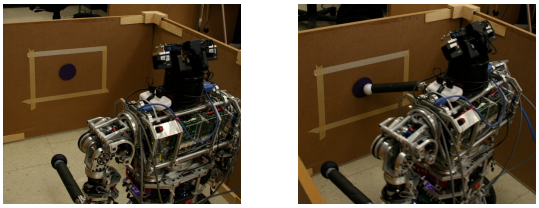
Figure 7. Learning curves in the PinBall domain, for agents employing skill trees created from demonstration trajectories, agents using incremental skill chaining, and agents starting with pre-learned skills.



(a) (b)



(c) (d)



(e) (f)

Figure 8. The task demonstrated on the uBot-5. Starting at the beginning of a corridor (a), the uBot approaches (b) and pushes open a door (c), turns through the doorway (d), then approaches (e) and pushes a panel (f).

To simplify perception, purple, orange and yellow colored circles were placed on the door and panel, beginning of the back wall, and middle of the back wall, respectively, as perceptually salient markers. The distances (obtained using onboard stereo vision) between the uBot to each marker were computed at 8Hz and filtered. The uBot was able to engage one of two motor abstractions at a time: either performing end-point position control of its hand, or controlling the speed and angle of its forward motion. Using these features, we constructed six sensorimotor abstractions, one for each pairing of salient object and motor command set. We performed policy regression to fit the segmented policies for replay. Policy replay testing was performed by varying the starting point of the robot by hand and used hand-coded stopping conditions that corresponded to the initiation set of the subsequent skill.

Of the 12 demonstration trajectories gathered from the uBot, 3 had to be discarded because of data loss due to excess perceptual noise. Of the remaining 9, all segmented sensibly and 8 were able to be merged into a single skill chain. Figure 9 shows a segmented trajectory obtained using CST, with Table 1 providing a brief description of the skills extracted along with their selected abstractions, and the number of sample trajectories required for each skill to be replayed successfully at least 9 times out of 10.

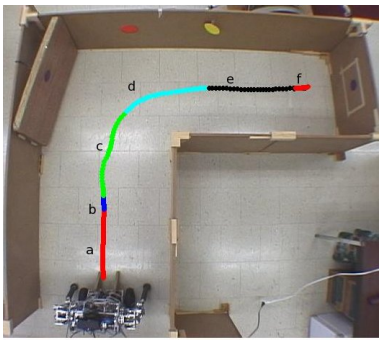


Figure 9. A demonstration trajectory of the uBot-5 performing the task in Figure 8, segmented into skills.

#	Abstraction	Description	Examples Required
a	torso-purple	Drive to door.	2
b	hand-purple	Open the door.	1
c	torso-orange	Drive toward wall.	1
d	torso-yellow	Turn toward panel.	2
e	torso-purple	Drive to the panel.	1
f	hand-purple	Press the panel.	3

Table 1. A brief description of each of the skills extracted from the trajectory shown in Figure 9, along with their selected abstractions, and the number of example trajectories required for accurate replay.

CST is thus able to successfully segment trajectories demonstrated on a mobile manipulator and assign the

relevant abstractions to each individual skill. Since each skill is defined using only a small number of relevant task variables, it requires a very low number of demonstration trajectories to achieve reliable replay. For details see Konidaris et al. (2010).

A similar experiment used CST in combination with model-based control methods for obtaining the skill policies. Here, the skill goals identified by CST (a small region around the first few states of the successor skill) were used as targets for closed-loop controllers. By segmenting the demonstrated trajectory into subtasks, we were able to achieve reliable replay using *just a single demonstration* with a simple control algorithm that would have been difficult or impossible to apply monolithically. For details see Kuindersma et al. (2010).

4.3. Acquiring Mobile Manipulation Skills from a Learned Policy

We now describe a robot system that produces its own demonstration trajectories by learning to sequence existing controllers and extracts skills from the resulting learned solution. In the Red Room task, the uBot-5 is placed in a small room containing a button and a handle. When the handle is pulled after the button has been pressed, a door in the side of the room opens, allowing the robot access to a compartment which contains a switch. The goal of the task is to press the switch. A schematic and photographs of the domain are given in Figure 10.

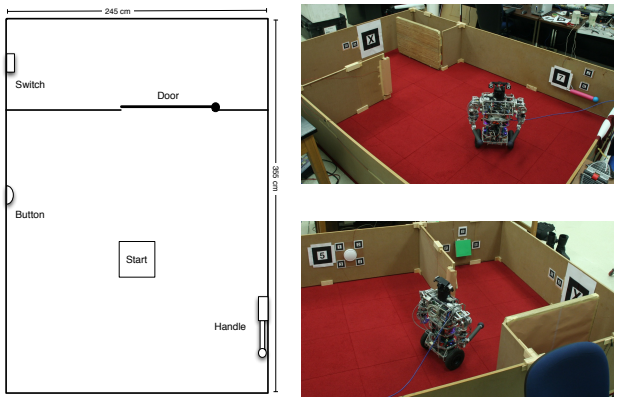


Figure 10. The Red Room Domain.

The robot is given a fixed set of innate controllers for navigating to visible objects of interest and for interacting with them using its end-effector (by extending its arm and moving it to the right, left, up, down, or forwards). In order to actuate the button and the switch, the robot must extend its arm and then move it outwards; in order to actuate the handle, it must extend its arm and then move it downwards. The robot constructed a transition model of the domain through interaction and used it to compute a policy using dynamic programming. The robot was able to find the optimal solution after five episodes.

The resulting optimal sequence of controllers was then used to generate 5 demonstration trajectories for use in CST. The resulting trajectories all segmented into the same sequence of 10 skills, and were all merged successfully. An example segmentation is shown in Figure 11; a description of each skill along with its relevant abstraction is given in Table 2.

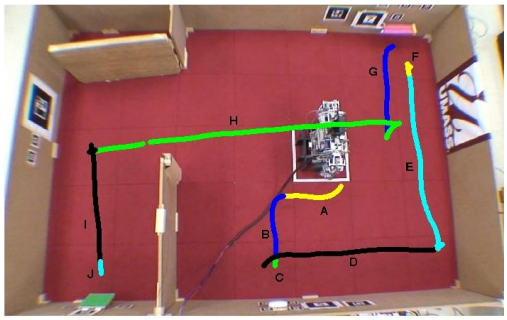


Figure 11. A trajectory from the learned solution to the Red Room task, segmented into skills.

#	Abstraction	Description
A	torso-button	Align with the button.
B	torso-button	Approach the button.
C	hand-button	Push the button.
D	torso-handle	Align with the handle.
E	torso-handle	Approach the handle.
F	hand-handle	Pull the handle.
G	torso-entrance	Align with the entrance.
H	torso-entrance	Drive through the entrance.
I	torso-switch	Approach the switch.
J	hand-switch	Press the switch.

Table 2. A brief description of each of the skills extracted from the trajectory shown in Figure 11, along with their selected abstractions.

CST consistently extracted skills that corresponded to manipulating objects in the environment, and navigating towards them. In the navigation case, each controller execution was split into two separate skills. These skills correspond exactly to the two phases of the navigation controller: first, aligning the robot with the normal of the target object, and second, moving the robot toward that feature. In the object-manipulation case, a sequence of two controllers is collapsed into a single skill: for example, extending the arm and then

extending it further toward the object of interest is collapsed into a single skill which we might label *push the button*. We constructed closed-loop manipulation policies by fitting splines over relative spatial waypoints and obtained reliable replay using a single demonstration trajectory for each. Once these skills were added to the robot's control system, it was able to deploy them in a second problem to solve it faster than it could using just its innate controllers. For details see Konidaris et al. (2011a).

5. Related Work

A great deal of work exists under the general heading of LfD (surveyed by Argall et al. (2009)). Most methods learn an entire policy monolithically from data, although some perform segmentation to extract skills.

The approach most closely related to CST is by Dixon and Khosla (2004a), where a demonstration trajectory is segmented into a sequence of linear dynamical systems. Each linear dynamical system is used to derive a convergent controller, with a small region around the final state considered its goal. The algorithm can be run online and was used in conjunction with several other methods to build a mobile robot system that performed LfD by tracking a human user (Dixon and Khosla, 2004b). This method differs from CST in three ways: it does not use skill-specific abstractions; it segments demonstration trajectories into policies that are linear in the robot's state variables, which is a stronger condition than a value function that is linear in a set of basis functions; and finally, it uses a heuristic method (an error metric exceeding a threshold parameter) for segmentation.

Another closely related LfD framework is the Performance-Derived Behavior Vocabularies (PDBV) framework (Jenkins and Matarić, 2004), which segments demonstrated data into motion primitives and clusters them to build a motion primitive library. CST differs from PDBV in four major aspects. First, PDBV is a batch method. Second, PDBV discovers a low-dimensional representation (a *motion-manifold*) for each primitive, rather than using an abstraction library. Third, PDBV extracts motion *policies* directly, which are not amenable to automatic improvement because they do not have goals. Finally, PDBV performs segmentation using Kinematic Centroid Segmentation, a heuristic specific to human-like motions. More recent work has used computationally expensive but principled statistical methods (Grollman and Jenkins, 2010; Butterfield et al., 2010) to segment the data as a way to avoid perceptual aliasing in the policy.

Chiappa et al. (2009) and Chiappa and Peters (2010) described a principled statistical approach to acquiring motion primitive libraries from data, where the demonstrated trajectories are modeled as Bayesian linear Gaussian state space models. This approach automatically segments the demonstrated data into policies, and it can handle multivariate target variables and models that repeat within a single trajectory. Although these systems have achieved impressive results on real robots, they use computationally intensive batch processing, do not use skill-specific abstractions, and do not result in skills with goals.

6. Discussion and Conclusion

The availability of a suitable abstraction library is key to the application of CST in high-dimensional domains. This could potentially require significant (though in principle once-off) design effort to create, program and debug a set of abstractions suitable for representing anything the robot may decide to learn. However, we expect that a small library consisting of pairings of motor abstractions and one or two visible objects will often be sufficient. If necessary, abstraction selection could be paired with a feature selection method to augment the selected abstraction during policy learning. Future work may also consider approaches to acquiring the abstraction library from data or finding feasible ways to build skill-specific abstractions at runtime.

An important assumption made by CST is that each skill segmentation should form a chain and the merged chains should form a tree. In some cases, however, they may form a more general graph (e.g., when the demonstrated policy has a loop). The procedure to merge skill chains could be generalized to accommodate such cases, and applied to merging skills both within an individual chain and across chains.

CST is agnostic to the method use to learn each individual skill policy from data. The experiments reported here used four different methods: value function regression in Pinball, both regression on motor output and model-based planning for acquiring policies from a human expert on the uBot, and a spline-based closed-loop controller for acquiring skills from learned controller sequences on the uBot. For robust and reliable robot control from demonstration data we expect that either a stabilized, trajectory-following controller or a dynamic movement primitive (Schaal, 2003) controller will provide a good balance between learnability and efficiency.

CST aims to extract transferrable policy components that the robot can retain, refine, and reuse. This is accomplished through a principled approach to learn-

ing from multiple unsegmented trajectories, and the use of skill-specific abstractions which enable efficient representation and facilitate transfer. These advantages offer a promising avenue of development toward general-purpose learning from demonstration.

Acknowledgments

We would like to thank the members of the LPR for their technical assistance. Andrew Barto and George Konidaris were supported in part by the AFOSR under grant FA9550-08-1-0418. George Konidaris was also supported in part by the AFOSR under grant AOARD-104135 and the Singapore Ministry of Education under a grant to the Singapore-MIT International Design Center. Scott Kuindersma is supported by a NASA GSRP fellowship from Johnson Space Center. Rod Grupen was supported by the Office of Naval Research under MURI award N00014-07-1-0749.

References

B. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57:469–483, 2009.

R.R. Burridge, A.A. Rizzi, and D.E. Koditschek. Sequential composition of dynamically dextrous robot behaviors. *International Journal of Robotics Research*, 18(6): 534–555, 1999.

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.

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.

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.

K.R. Dixon and P.K. Khosla. Trajectory representation using sequenced linear dynamical systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3925–3930, 2004a.

K.R. Dixon and P.K. Khosla. Learning by observation with mobile robots: a computational approach. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 102–107, 2004b.

P. Fearnhead and Z. Liu. On-line inference for multiple changepoint problems. *Journal of the Royal Statistical Society B*, 69:589–605, 2007.

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

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.

G.D. Konidaris and A.G. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems 22*, pages 1015–1023, 2009.

G.D. Konidaris, S.R. Kuindersma, A.G. Barto, and R.A. Grupen. Constructing skill trees for reinforcement learning agents from demonstration trajectories. In J. Lafferty, C.K.I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1162–1170, 2010.

G.D. Konidaris, S.R. Kuindersma, R.A. Grupen, and A.G. Barto. Autonomous skill acquisition on a mobile manipulator. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*, 2011a.

G.D. Konidaris, S. Osentoski, and P.S. Thomas. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*, 2011b.

S.R. Kuindersma, G.D. Konidaris, R.A. Grupen, and A.G. Barto. Learning from a single demonstration: Motion planning with skill segmentation. In *Proceedings of the NIPS Workshop on Learning and Planning from Batch Time Series Data*, December 2010.

L-J Lin. Programming robots using reinforcement learning and teaching. In *Proceedings of the Ninth National conference on Artificial Intelligence*, pages 781–786, 1991.

T. Lozano-Perez, M.T. Mason, and R.H. Taylor. Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research*, 3(1):3–24, 1984.

F.G. Martin. *The Handy Board Technical Reference*. MIT Media Lab, Cambridge MA, 1998.

S. Schaal. Dynamic movement primitives - a framework for motor control in humans and humanoid robots. In *Proceedings of the international symposium on adaptive motion of animals and machines*, 2003.

R.S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063, 2000.

R. Tedrake. LQR-Trees: Feedback motion planning on sparse randomized trees. In *Proceedings of Robotics: Science and Systems*, pages 18–24, 2009.