Article

The International Journal of
Robotics Research
1–20

DOI: 10.1177/0278364918784350
journals.sagepub.com/home/ijr

SWIRL: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards

Sanjay Krishnan[1], Animesh Garg[1,2], Richard Liaw[1], Brijen Thananjeyan[1], Lauren Miller[1], Florian T Pokorny[3] and Ken Goldberg[1]

Abstract

We present sequential windowed inverse reinforcement learning (SWIRL), a policy search algorithm that is a hybrid of exploration and demonstration paradigms for robot learning. We apply unsupervised learning to a small number of initial expert demonstrations to structure future autonomous exploration. SWIRL approximates a long time horizon task as a sequence of local reward functions and subtask transition conditions. Over this approximation, SWIRL applies Q-learning to compute a policy that maximizes rewards. Experiments suggest that SWIRL requires significantly fewer rollouts than pure reinforcement learning and fewer expert demonstrations than behavioral cloning to learn a policy. We evaluate SWIRL in two simulated control tasks, parallel parking and a two-link pendulum. On the parallel parking task, SWIRL achieves the maximum reward on the task with 85% fewer rollouts than Q-learning, and one-eight of demonstrations needed by behavioral cloning. We also consider physical experiments on surgical tensioning and cutting deformable sheets using a da Vinci surgical robot. On the deformable tensioning task, SWIRL achieves a 36% relative improvement in reward compared with a baseline of behavioral cloning with segmentation.

Keywords

Reinforcement learning, inverse reinforcement learning, learning from demonstrations, medical robots and systems

1. Introduction

There has been significant recent interest in learning control policies through autonomous exploration (Agrawal et al., 2016; Finn and Levine, 2017; Levine et al., 2016; Pinto and Gupta, 2016; Pinto et al., 2016). However, it is difficult to scale such techniques to learning tasks with longer time horizons. Discovering a viable policy purely through exploration can be extremely difficult, especially if there are narrow criteria for task success. In long tasks, it can be very difficult to quantify the advantage of taking a particular action without a large number of repeated trials. Conversely, learning from demonstrations presents an alternative, where the robot imitates demonstrations collected from an expert supervisor (Argall et al., 2009). However, this places a significant burden on the supervisor to exhaustively cover the scenarios the robot may encounter during execution (Laskey et al., 2017). Moreover, in the learning from demonstrations paradigm, the robot can, at best, only match the performance of the demonstrator.

To address the limitations of either extreme, this article proposes a hybrid of the exploration and demonstration learning paradigms. We apply unsupervised learning to a small number of initial expert demonstrations to structure future autonomous exploration. A real-world task often naturally decomposes into a sequence of simpler, locally solvable subtasks. For example, an assembly task might decompose into completing the part's constituent subassemblies or a surgical task might decompose into a sequence of movement primitives. Such a structure imposes a strong prior on the class of successful policies and can focus exploration in reinforcement learning. It reduces the effective time horizon of learning to the start of the next subtask rather than until task completion. We apply a clustering algorithm to identify a latent set of state-space subgoals that

[1]AUTOLAB, University of California, Berkeley, USA
[2]Stanford University, USA
[3]RPL/CSC, KTH Royal Institute of Technology, Sweden

Corresponding author:
Sanjay Krishnan, AUTOLAB, University of California, Berkeley, California 94720, USA.
Email: sanjay@eecs.berkeley.edu

sequentially combine to form a global task. This leads to a novel policy search algorithm, called sequential windowed inverse reinforcement learning (SWIRL), where the demonstrations can bootstrap a self-supervised Q-learning algorithm.

The unsupervised learning model is based on our prior work on transition state clustering (Krishnan et al., 2015, 2016; Murali et al., 2016). Transitions are defined as significant changes in the state trajectory. These transitions can be spatially and temporally clustered to identify whether there are common conditions that trigger a change in motion across demonstrations. The algorithm is outlined in Algorithm 1. SWIRL extends this basic model with an inverse reinforcement learning step that extracts subgoals and computes local cost functions from the learned clusters. Learning a policy over the segmented task is nontrivial because solving k independent problems neglects any shared structure in the value function during the policy learning phase (e.g., a common failure state). Jointly learning over all segments introduces a dependence on history, namely, any policy must complete step i before tackling step $i+1$. Learning a memory-dependent policy could lead to an exponential overhead of additional states. We show that the problem can be posed as a proper Markov decision process in a lifted state space that includes an indicator variable of the highest-index $\{1, \ldots, k\}$ transition region that has been reached so far if there are Markovian regularity assumptions on the clustering algorithm.

The problem setting in SWIRL subtly differs from what is typical in imitation learning and other works on learning from demonstrations. During the demonstration phase, SWIRL only observes the sequence of states visited by the supervisor, and does not need to know what action the supervisor applied or require a model of the environment dynamics. We believe that this makes SWIRL applicable in the increasingly popular third-person imitation learning settings, where robots can exploit large corpora of videos of human beings or other robots performing a variety of tasks (Stadie et al., 2017).

This article is a significantly revised and extended version of a previous conference publication (Krishnan et al., 2016) with new experiments, formalism, and analysis. In particular, this article introduces a new class of unsupervised trajectory segmentation algorithms that can be efficiently paired with reinforcement learning, as they preserve Markovian structure. This paper also extends the number of baseline approaches considered in simulation and adds a new experiment in using a da Vinci surgical robot to manipulate deformable materials.

The content is structured as follows. Section 2 describes the background and related work. Section 3 describes notation and the problem statement. Section 4 gives an overview of the entire SWIRL framework. Section 5 describes the clustering algorithm applied to the initial demonstrations to divide the task into consistent subtasks. Section 6 describes how local reward functions are assigned to each of the subtasks to construct a segmented approximation of the global task. Section 7 describes how to apply reinforcement learning to learn a policy over this approximation. Finally, Sections 8 to 10 discuss the experimental methodology and results.

2. Related work and background

2.1. Apprenticeship learning

Abbeel and Ng (2004) argue that the reward function is often a more concise representation of a task than a policy is. As such, a concise reward function is more likely to be robust to small perturbations in the task description. The downside is that the reward function is not useful on its own and, ultimately, a policy must be retrieved. In the most general case, a reinforcement learning algorithm must be used to optimize for that reward function (Abbeel and Ng, 2004). It is well-established that reinforcement learning problems often converge slowly in complex tasks when rewards are sparse and not "shaped" appropriately (Judah et al., 2014; Ng et al., 1999). Our work revisits this two-phase algorithm in the context of sequential tasks and techniques, in order to scale such an approach to longer time horizons. Related to SWIRL, Kolter et al. (2007) studied *hierarchical apprenticeship learning* to learn bipedal locomotion, where the algorithm is provided with a hierarchy of subtasks. We explore the automatical inference of a sequential task structure from data.

2.2. Motion primitives

Researchers in learning from demonstrations and planning have studied the problem of leveraging motion primitives, or libraries of temporally extended action sequences, to improve generalization. Dynamic motion primitives are used to construct new motions through a composition of dynamic building blocks (Ijspeert et al., 2002; Manschitz et al., 2015; Pastor et al., 2009). Much of the work in motion primitives has considered manually identified segments but Niekum et al. (2012) proposed learning the set of primitives from demonstrations using the Beta-Process Autoregressive Hidden Markov Model. Similarly, Calinon (2014) proposed a task-parametrized movement model with Gaussian mixture models for automatic action segmentation. Both Niekum et al. (2012) and Calinon (2014) considered the motion planning setting in which analytical planning methods are used to plan and perform a task.

2.3. Segmentation

Trajectory segmentation is a well-studied area of research, dating back to early biomechanics and robotics research. For example, Viviani and Cenzato (1985) explored using the "two-thirds" power law coefficient to determine segment boundaries in handwriting. Morasso (1983) showed that rhythmic three-dimensional motions of a human arm could be modeled as piecewise linear. In a seminal paper, Sternad and Schaal (1999) provided a formal framework for

the control-theoretic segmentation of trajectories. Botvinick et al. (2009) explored the reinforcement learning analog of the control-theoretic models. Concurrently, temporal segmentation was being developed by researchers in motion capture (Moeslund and Granum, 2001). Recently, some Bayesian approaches have been proposed for the segmentation problem (Asfour et al., 2008; Calinon and Billard, 2004; Kruger et al., 2010; Tanwani and Calinon, 2016; Vakanski et al., 2012). Two challenges are those of collecting enough data to employ these techniques and of tuning the hyperparameters. In a prior work, we observed that, under the assumption that the task is sequential (with the same order of primitives in each demonstration), the inference could be modeled as a two-level clustering problem (Krishnan et al., 2015). This results in improved accuracy and robustness for a small number of demonstrations. Another relevant result is that of Ranchod et al. (2015), who use an inverse reinforcement learning model to define the primitives, in contrast with the problem of learning a policy after inverse reinforcement learning.

2.4. Hierarchical reinforcement learning

The field of hierarchical reinforcement learning has a long history in artificial intelligence (Barto and Mahadevan, 2003; Sutton et al., 1999) and in the analysis of biological systems (Botvinick, 2008; Botvinick et al., 2009; Solway et al., 2014; Whiten et al., 2006; Zacks et al., 2011). Early work in hierarchical control demonstrated the advantages of hierarchical structures by handcrafting hierarchical policies (Brooks, 1986) and learning them, given various manual specifications: state abstractions (Dayan and Hinton, 1992; Hengst, 2002; Kolter et al., 2007; Konidaris and Barto, 2007), a set of waypoints (Kaelbling, 1993), low-level skills (Bacon and Precup, 2015; Huber and Grupen, 1997), a set of finite-state meta-controllers (Parr and Russell, 1997), a set of subgoals (Dietterich, 2000; Sutton et al., 1999), or intrinsic reward (Kulkarni et al., 2016). The key abstraction used in hierarchical reinforcement learning is the "options" framework, where subskills are represented by local policies, termination conditions, and initialization conditions. A high-level policy switches between these options and combines them into a larger task policy. In this framework, per-subskill reward functions are called subgoals. SWIRL is an algorithm for learning quadratic subgoals and termination conditions, where the high-level policy is deterministic and sequentially iterates through the subskills.

3. Problem setup

A finite-horizon Markov decision process is a five-tuple $\langle S, A, P(\cdot\,,\cdot), T, \mathbf{R}\rangle$, where S is the set of states (continuous or discrete), A is the set of actions, $P : S \times A \mapsto \Delta(S)$ is the dynamic model that maps states and actions to a probability density over subsequent states, T is the time horizon, and $\mathbf{R} : S \times A \mapsto \mathbb{R}$ is the reward function.

3.1. Problem setting

We assume access to a supervisor, who samples from an unknown policy $\pi : S \mapsto \Delta(A)$ that maps states to a probability distribution over actions. These samples form sequences of length T, called demonstrations. The goal of these demonstrations is to provide an initial dataset from which we can learn about the structure of the Markov decision process. In the most basic form, a demonstration is defined as follows.

Definition 1. Fully observed demonstration. A demonstration $\mathbf{d}$ is a sequence of tuples of states and actions

$$\mathbf{d} = [(s_0, a_0), (s_1, a_1), \ldots, (s_T, a_T)]$$

This fully observed setting is typical of the literature on imitation learning (Osa et al., 2018), where one uses a function approximation to learn the state to action mapping from these samples. However, one may have a more limited access to a limited supervisor, where only the states are observed. The actions that the supervisor applied are latent.

Definition 2. State demonstration. A state demonstration d is a sequence of tuples of states and actions

$$d = [(s_0, \_), (s_1, \_), \ldots, (s_T, \_)]$$

The state demonstration setting differs from what is classically studied in imitation learning. This can happen when the demonstration modality differs from the execution setting, e.g., learning from third-person videos, motion capture of a human being performing a task, or kinesthetic demonstrations. Our objective is a framework that can learn even if only state demonstrations are provided. Notationally, we will use **boldface** to denote fully observed demonstrations and the regular font face to denote state demonstrations.

Problem 1. Policy search. Given a Markov decision process and a set of state demonstration trajectories $D = \{d_1, \ldots, d_N\}$ from a supervisor, return a policy $\pi : S \mapsto \Delta(A)$ that maximizes the cumulative reward of the Markov decision process.

4. Task model

Directly optimizing the reward function $\mathcal{R}$ in the Markov decision process from the previous section might be very challenging. We propose to approximate $\mathcal{R}$ as a sequence of smoother reward functions.

Definition 3. Proxy task. A proxy task is a set of k Markov decision processes with the same state set, action set, and dynamics. Associated with each Markov decision process, i, is a reward function $R_i : S \times A \mapsto \mathbb{R}$. Additionally, associated with each R_i is a transition region $\rho_i \subseteq S$, which is a subset of the state space. A robot accumulates a reward R_i until it reaches the transition region ρ_i, then the robot switches to the next reward and transition pair. This process continues until ρ_k is reached.

A robot is deemed *successful* when all of the ρ_i are reached in sequence within a global time horizon T. SWIRL uses a set of initial supervisor demonstrations to construct a proxy task that approximates the original Markov decision process.

4.1. Modeling assumptions

To make this problem computationally tractable, we make some modeling assumptions.

Modeling assumption 1. Successful demonstrations. We need conditions on the demonstrations to be able to infer the sequential structure. We assume that all demonstrations are successful, that is, they visit each ρ_i in the same sequence.

Modeling assumption 2. Quadratic rewards. We assume that each reward function R_i can be expressed as a quadratic of the form $-(s-s_0)^T\Psi(s-s_0)$ for some positive semi-definite Ψ and a center point s_0 with $s_0^T\Psi s_0 = 0$.

Modeling assumption 3. Ellipsoidal approximation. Finally, we assume that the transition regions in ρ_i can be approximated as a set of disjoint ellipsoids.

4.2. Algorithm overview

SWIRL can be described in terms of three subalgorithms:
Inputs: Demonstrations D

1. **Sequence learning (Section 5)**. Given D, SWIRL applies a hierarchical clustering algorithm to partition the task into k subtasks, whose starts and ends are defined by arrival at a sequence of transitions $G = [\rho_1, \ldots, \rho_k]$.
2. **Reward learning (Section 6)**. Given G and D, SWIRL associates a local reward function with each segment, resulting in a sequence of rewards $\mathbf{R}_{\text{seq}} = [R_1, \ldots, R_k]$.
3. **Policy learning (Section 7)**. Given $\mathbf{R}_{\text{seq}}$ and G, SWIRL applies reinforcement learning to optimize a policy for the task π.

Outputs: Policy π

In principle, one could couple steps 1 and 2, as with the results of Ranchod et al. (2015). We separate these steps, since this allows us to use a different set of features for segmentation than is used for reward learning. Perceptual features can provide a valuable signal for segmentation but quadratic reward functions might not be meaningful in all perceptual feature spaces.

5. Sequence learning

First, SWIRL applies a clustering algorithm to the initial demonstrations to learn the transition regions. The clustering model is based on our prior work on transition state clustering (Krishnan et al., 2015; Murali et al., 2016). Transitions are defined as significant changes in the state trajectory. These transitions can be spatially and temporally clustered to identify whether there are common conditions that trigger a change in motion across demonstrations.

5.1. General framework

In a first pass, the clustering algorithm applies a motion-based segmentation model over the noisy trajectories and identifies a set of candidate segment transitions in each.

Definition 4. Transition indicator function. A transition indicator function $\mathbf{T}$ is a function that maps each time index in a demonstration d to $\{0, 1\}$.

As in Lioutikov et al. (2015), this is an initial heuristic that oversegments to ground the probabilistic segmentation. One can apply a variety of different change point detection methodologies, such as detection of significant changes in image features (Murali et al., 2016) or dynamic transitions (Niekum et al., 2012), or the use of Gaussian mixture models (Krishnan et al., 2015). The particular model chosen depends on the perceptual features available.

This function marks candidate segment endpoints, called transitions, in a trajectory. Definition 4 naturally leads to a notion of transition states, that is, the states and times at which transitions occur.

Definition 5. Transition states. For a demonstration set D, transition states are the set of state-time tuples where the indicator is 1

$$\Gamma = \{(x,t) \in D : \mathbf{T}(x) = 1\}$$

We model the set Γ as samples from an underlying distribution over the state space and time

$$\Gamma \sim f(x,t)$$

We model this distribution using a Gaussian mixture model

$$f(x,t) \approx \text{GMM}(\eta, \{\mu_1, \ldots, \mu_k\}, \{\Sigma_1, \ldots, \Sigma_k\})$$

The interpretation of this distribution is that η describes the fraction of transitions assigned to each mixture component, μ_i describes the centroid of the mixture component, and Σ describes the covariance. While Gaussian mixture models are a poor approximation for some distributions, they have shown empirical success for trajectory segmentation (Calinon, 2014). Our prior work (Krishnan et al., 2015; Murali et al., 2016) describes a number of practical optimizations, such as the pruning of low-confidence mixture components.

For each mixture component, we can define ellipsoids by taking the confidence level sets in the state space and time that characterize regions where transitions occur. These regions are ordered, as they are also defined over time, since we make the assumption that the confidence threshold for the level sets is tuned so that the regions are

disjoint. Thus, each of these regions defines a testable condition based on the current state, time, and previously reached regions—which is a Markov segmentation function. The result is exactly the set of transition regions, $G = [\rho_1, \rho_2, \ldots, \rho_k]$, and segmentation of each demonstration trajectory into k segments.

In typical Gaussian mixture model formulations, one must specify the number of mixture components k beforehand. However, we apply results from Bayesian nonparametric statistics and jointly solve for the component parameters and the number of components using a Dirichlet process (Kulis and Jordan, 2012). The Dirichlet process places a soft prior on the number of clusters. During inference, the number of components grows with the complexity of the observed data (we denote this "DP-GMM"). The Dirichlet process has hyperparameters, which we tune once for all domains; we use a uniform base measure and a prior weight of 0.1.

5.2. Properties

While there are several different algorithms for segmenting a set of demonstrations into subsequences, not all are directly applicable to this problem setting. In our problem, segments are used to construct a proxy Markov decision process task for the robot to optimize. During the offline phases of the algorithm (sequence learning and reward learning), the clustering algorithm observes the full demonstration trajectory. However, during the learning phase (policy learning), the algorithm only observes the partial trajectory up to the current time-step and does not observe which segment is active. In this sense, segmentation introduces a problem of partial observation, even if the original task is fully observed. The segmentation must be estimated from the history of the process.

Trivially, some algorithms are not applicable, since they might require knowledge of future data. Even if the algorithm is causal, it might have an arbitrary dependence on the past, leading to inefficient estimation of the currently active segment. To address this problem, we formalize the following condition.

Definition 6. Markov segmentation. A segmentation of a task is a function F that maps every state-time tuple to an index $\{1, \ldots, k\}$

$$F : S \times \mathbb{Z}_+ \mapsto \{1, \ldots, k\}$$

A Markov segmentation function is a task segmentation, where the segmentation index at time $t+1$ can be completely determined by the featurized state s_t at time t and the index i_t at time t

$$i_{t+1} = \mathbf{M}(s_t, i_t)$$

5.3. Transition indicator model

The general framework can support a number of different transition indication heuristics. This article focuses on a mixture modeling approach that identifies when trajectories leave certain clusters of motion primitives. This technique is quite general and applies to a large class of systems with low-dimensional state spaces. Refer to our prior work for other variations (Krishnan et al., 2015).

Algorithm 1: Subgoal clustering.

Data: Demonstration $\mathcal{D}$
1 Use a transition indicator heuristic to identify a set of candidate transitions Θ.
2 Fit a DP-GMM to the states in Θ.
3 Prune clusters that do not have one transition from all demonstrations.
4 The result is $G = [\rho_1, \rho_2, \ldots, \rho_m]$, where each ρ is a disjoint ellipsoidal region of the state space and time interval.
Result: G

For a given time t, we can define a window of length ℓ as

$$\mathbf{w}_t^{(\ell)} = [s_{t-\ell}, \ldots, s_t]^\mathsf{T}$$

We can further normalize this window relative to its first state, as

$$\mathbf{n}_t^{(\ell)} = [s_{t-\ell} - s_{t-\ell}, \ldots, s_t - s_{t-\ell}]^\mathsf{T}$$

This represents the "delta" in movement over the time span of a window. Then, for each demonstration trajectory, we can also generate a trajectory of $T - \ell$ windowed states, as

$$\mathbf{d}^{(\ell)} = [\mathbf{n}_\ell^{(\ell)}, \ldots, \mathbf{n}_T^{(\ell)}]^\mathsf{T}$$

Over the entire set of windowed demonstrations, we collect a dataset of all of the $\mathbf{n}_t^{(\ell)}$ vectors. We fit a Gaussian mixture model to these vectors. The Gaussian mixture model defines m multivariate Gaussian distributions and a probability that each observation $\mathbf{n}_t^{(\ell)}$ is sampled from each of the m distributions. We annotate each observation with the most likely mixture component. Times such that $\mathbf{n}_t^{(\ell)}$ and $\mathbf{n}_{t+1}^{(\ell)}$ have different most-likely components are marked as transitions. This model captures some dynamic behaviors while not requiring explicit modeling of the state-to-state transition function.

Sometimes the Markov decision process's states are more abstract and do not map to space where the normalized windows make sense. We can still apply the same method when we only have a positive definite kernel function over all pairs of states $\kappa(s_i, s_j)$. We can construct this kernel function for all of the states observed in the demonstrations and apply kernelized principle component analysis to the features before learning the transitions—a technique used in computer vision (Mika et al., 1998). The top p' eigenvalues define a new embedded feature vector for each ω in $\mathbb{R}^{p'}$. We can now apply Algorithm 5.3 in this embedded feature space.

6. Reward learning

After the sequence learning phase, each demonstration is partitioned into k segments. The reward learning phase uses the learned transition regions $[\rho_1, \ldots, \rho_k]$ to construct the local rewards $[R_1, \ldots, R_k]$ for the task. Each R_i is a quadratic cost parametrized by a positive semi-definite matrix Ψ.

Inverse reinforcement learning (Ng and Russell, 2000) involves studying the problem of inferring an implicitly optimized reward function from demonstrations. The literature on inverse reinforcement learning argues that the reward function is often a more concise representation of a task than a policy is. As such, a concise reward function is more likely to be robust to small perturbations in the task description. The problem of constructing reward functions is very much related to this problem. The only caveat is that most inverse reinforcement learning techniques require a dynamic model (Ziebart et al., 2008) or fit a local dynamic model (Finn et al., 2016). Fitting a local dynamic model is not possible with state demonstrations, as we do not explicitly observe the demonstrators' actions.

6.1. Method for state-only demonstrations

The role of the reward function is to guide the robot to the next transition region ρ_i. A first approach is that, for each segment i, we can define a reward function as

$$R_i(s) = - \parallel s - \mu_i \parallel_2^2$$

which is simply the Euclidean distance from the centroid.

A problem in using the Euclidean distance directly is that it uniformly penalizes disagreement with μ in all dimensions. During different stages of a task, some directions are naturally likely to vary more than others. To account for this, we can derive

$$\Psi[j,l] = \Sigma^{-1}$$

which is the inverse of the covariance matrix of all of the state vectors in the segment

$$\Psi = \left(\sum_{t=\text{start}}^{\text{end}} ss^{\mathsf{T}} \right)^{-1} \tag{1}$$

which is a $p \times p$ matrix defined as the covariance of all of the states in the segment $i-1$ to i. Intuitively, if a feature has low variance during this segment, deviation in that feature from the desired target is penalized. This is exactly the Mahalanobis distance to the next transition.

For example, suppose that one of the features j measures the distance to a reference trajectory u_t. Further, suppose that in step one of the task the demonstrator's actions are perfectly correlated with the trajectory ($\Psi_i[j,j]$ is low where variance is in the distance) and in step two the actions are uncorrelated with the reference trajectory ($\Psi_i[j,j]$ is high). Thus, Ψ will penalize deviation from $\mu_i[j]$ more in step one than in step two.

6.2. Method for full demonstrations

If we do have access to full demonstrations, we can locally fit dynamic models and use a standard inverse reinforcement learning technique. In our experiments, we compare both approaches—and find that the covariance-based method given here is actually very competitive.

6.2.1. Primer on maximum entropy inverse reinforcement learning. Maximum entropy inverse reinforcement learning (MaxEnt-IRL) (Ziebart et al., 2008) involves finding a reward function such that an optimal policy with respect to the reward function is close to the expert demonstration. In the MaxEnt-IRL model, "close" is defined as matching the first moment of the expert feature distribution

$$\gamma_{\text{expert}} = \frac{1}{Z} \sum_{d \in D} \sum_{i=1}^{N} s_i$$

where Z is an appropriate normalization constant (the total number of states in all demonstrations). The MaxEnt-IRL model uses the linear parametrized representation

$$R(x) = s^{\mathsf{T}} \theta$$

where x is a feature vector representing the state of the system. The agent is modeled as noisily optimal, where it takes actions from a policy π

$$\pi(a|s,\theta) \propto \exp\{A_\theta(s,a)\}$$

where A_θ is the advantage function (the Q function minus the value function) for the reward, parameterized by θ. The objective is to maximize the log-likelihood that the demonstration trajectories were generated by θ.

Under the exponential distribution model, it can be shown that the gradient for this likelihood optimization is

$$\frac{\partial L}{\partial \theta} = \gamma_{\text{expert}} - \gamma_\theta$$

where γ_θ is the first moment of the feature distribution of an optimal policy under θ.

SWIRL applies the MaxEnt-IRL model to each segment of the task but with a small modification; to learn quadratic rewards instead of linear ones. Let μ_i be the centroid of the next transition region. We want to learn a reward function of the form

$$R_i(x) = -(s-\mu_i)^{\mathsf{T}} \Psi (s-\mu_i)$$

for a positive semi-definite Ψ (negated, since this is a negative quadratic cost). With some re-parametrization (dropping μ_i for convenience and without loss of generality), this reward function can be written as

$$R_i(s) = -\sum_{j=1}^{d} \sum_{l=1}^{d} \Psi[i,j] s[j] s[l]$$

which is linear in the feature space $y = s[j]s[l]$

$$R_i(s) = \theta^{\mathsf{T}} y$$

6.2.2. Two inference settings: discrete and continuous. In the MaxEnt-IRL model, the gradient can be estimated reliably in two cases, discrete and linear Gaussian systems, since it requires an efficient forward search of the policy, given a particular reward parametrized by θ. In both these cases, we have to estimate the system dynamics within each segment.

Consider the case when the state space is discrete and the action space is discrete. To estimate the dynamics, we construct an $|S| \times |S| \times |A|$ matrix of zeros for each action, where each of the components of this matrix corresponds to the transition probability of a pair of states. For each, (s, a, s') observation in the segment, we increment $(+1)$ the appropriate element of the matrix. Finally, we normalize the elements to sum to one across the set of actions. An additional optimization could be to add smoothing to this estimate (i.e., initialize the matrix with some nonzero constant value); we found that this was not necessary for the sparse domains in our experiments. The result is an estimate of $P(s'|s, a)$. Given this estimate, γ_θ can be efficiently calculated with the forward-backward technique described by Ziebart et al. (2008).

The discrete model is difficult to scale to continuous state spaces. If we discretize, the number of bins required would be exponential in the dimensionality. However, linear models are another class of dynamic model for which the estimation and inference is tractable. We can fit local linear models to each of the segments discovered in the previous section

$$A_j = \arg\min_A \sum_{i=1}^{N} \sum_{\text{seg } j \text{ start}}^{\text{seg } j \text{ end}} \left\| A s_t^{(i)} - s_{t+1}^{(i)} \right\|$$

With A_j known, γ_θ can be analytically solved with techniques proposed by Ziebart et al. (2012). SWIRL applies MaxEnt-IRL to the subsequences of demonstrations between 0 and ρ_1, and then from ρ_1 to ρ_2, and so on. The result is an estimated local reward function R_i modeled as a linear function of states that is associated with each transition region ρ_i (Algorithm 2).

Algorithm 2: Reward inference.

Data: Demonstration $\mathcal{D}$ and subgoals $[\rho_1, \ldots, \rho_k]$
1 Based on the transition states, segment each demonstration d_i into k subsequences where the jth is denoted $d_i[j]$.
2 Apply MaxEnt-IRL or equation (1) to each set of subsequences $1, \ldots, k$.
Result: $\mathbf{R}_{\text{seq}}$

7. Policy learning

SWIRL uses the learned transition regions $[\rho_1, \ldots, \rho_k]$ and $\mathbf{R}_{\text{seq}}$ to construct a proxy task to solve via reinforcement learning. In this section, we describe a method of learning a policy π, given rewards $\mathbf{R}_{\text{seq}}$ and an ordered sequence of transitions G. However, this problem is not trivial, since solving k independent problems neglects potential shared value structures between local problems (e.g., a common failure state). Furthermore, simply taking the aggregate of the rewards can lead to inconsistencies, since there is nothing enforcing the order of operations. We show that a single policy can be learned jointly over all segments over a modified problem where the state space, with additional variables, keep tracks of the previously achieved segments. We present a Q-learning algorithm (Mnih et al., 2015; Sutton and Barto, 1998) that captures the noted coupling between task segments. In principle, similar techniques can be used for any other policy search method.

7.1. Jointly learning over all segments

In our sequential task definition, we cannot transition to reward R_{i+1} unless all previous transition regions $\rho_1, \ldots, \rho_i$ are reached in sequence. We can leverage the definition of the Markov segmentation function formalized earlier to learn jointly across all segments, while leveraging the segmented structure. We know that the reward transitions (R_i to R_{i+1}) only depend on an arrival at the transition state ρ_i and not any other aspect of the history. Therefore, we can store an index v, which indicates whether a transition state $i \in 0, \ldots, k$ has been reached. This index can be efficiently incremented when the current state $s \in \rho_{i+1}$. The result is an augmented state space $\begin{pmatrix} s \\ v \end{pmatrix}$ to account for previous progress. In this lifted space, the problem is a fully observed Markov decision process. Then the additional complexity of representing the reward with history over $S \times [k]$ is only $\mathcal{O}(k)$ instead of exponential in the time horizon.

7.2. Segmented Q-learning

At a high level, the objective of standard Q-learning is to learn the function $Q(s, a)$ of the optimal policy, which is the expected reward that the agent will receive on taking action a in state s, assuming that future behavior is optimal. Q-learning works by first initializing a random Q function. Then it samples rollouts from an exploration policy collecting (s, a, r, s') tuples. From these tuples, one can calculate

$$y_i = R(s, a) + \arg\max_a Q(s', a)$$

Each of the y_i can be used to define a loss function since, if Q were the true Q function, then the following recurrence would hold

$$Q(s, a) = R(s, a) + \arg\max_a Q(s', a)$$

Algorithm 3: Q-learning with segments.

Data: Transition states G, reward sequence $\mathbf{R}_{\text{seq}}$, exploration policy π
Initialize $Q\left(\begin{pmatrix} s \\ v \end{pmatrix}, a\right)$ randomly
foreach iter $\in 0, \ldots, I$ **do**
  Draw s_0 from initial conditions
  Initialize v to be $[0, \ldots, 0]$
  Initialize j to be 1
  foreach $t \in 0, \ldots, T$ **do**
    Choose action a based on π.
    Observe reward R_j
    Update state to s' and Q via Q-learning update
    If s' is $\in \rho_j$ update $v[j] = 1$ and $j = j + 1$
Result: Policy π

Thus, Q-learning defines a loss

$$L(Q) = \sum_i \| y_i - Q(s, a) \|_2^2$$

This loss can be optimized with gradient descent. When the state and action space is discrete, the representation of the Q function is a table, and we get the familiar Q-learning algorithm (Sutton and Barto, 1998)—where each gradient step updates the table with the appropriate value. When the Q function must be approximated, we get the deep Q-network algorithm (Mnih et al., 2015).

SWIRL applies a variant of Q-learning to optimize the policy over the sequential rewards. This is summarized in Algorithm 3. The basic change to the algorithm is that the state space is augmented with an indicator vector that indicates the transition regions that have been reached. So each of the rollouts now records a tuple $(s, \mathbf{v}, a, r, s', \mathbf{v}')$ that additionally stores this information. The Q function is now defined over states, actions, and segment index—which also selects the appropriate local reward function

$$Q(s, a, v) = R_v(s, a) + \arg\max_a Q(s', a, v')$$

We also need to define an exploration policy, i.e., a stochastic policy with which we will collect rollouts. To initialize the Q-learning, we apply behavioral cloning locally for each of the segments to get a policy π_i. We apply an ε-greedy version of these policies to collect rollouts.

8. Experimental methodology

This section overviews our basic experimental methodology. We evaluate SWIRL using two simulated reinforcement learning benchmarks and two deformable manipulation tasks on the da Vinci surgical robot (Kazanzides et al., 2014). In all of these tasks, there is a single Markov decision process of interest where the reward is sparse (for a substantial amount of the state space, the reward is zero). The goal of SWIRL is to improve convergence on this task.

8.1. Supervision

In both of the reinforcement learning benchmarks, the reward function defines a target configuration of the robot. We generated the initial demonstrations using an RRT* motion planner (assuming deterministic dynamics) (Karaman and Frazzoli, 2010). As both reinforcement learning benchmarks are stochastic in nature, we used a model-predictive-control-style re-planning approach to control the robot to the target region. For the physical experiments, we provided the robot with demonstrations collected through tele-operation. We collected fully observed trajectories with both states and actions.

8.2. Basic baselines

8.2.1. Pure exploration. The first set of baselines study a pure exploration approach to learning a policy. The baseline approach applies reinforcement learning to the global task. In all of our experiments, we use variants of Q-learning with different function approximators. The Q function is randomly initialized and is updated with data collected from episodic rollouts. The hyperparameters for each experiment are listed in the appendixes. We use approximate Q-learning (Bertsekas and Tsitsiklis, 1995; Thrun and Schwartz, 1993) in the simulated benchmarks and deep Q-networks in the physical experiments (Mnih et al., 2015).

8.2.2. Pure demonstration. This baseline directly learns a policy from an initial set of demonstrations using supervised learning. This approach is called behavioral cloning (see the survey of imitation learning given by Osa et al. (2018)); in each of our experiments, we describe the policy models used. It is important to note that this approach requires fully observed demonstrations.

8.2.3. Warm-start exploration. Next, we consider approaches that leverage both demonstrations and exploration. One approach is to use demonstrations to initialize the Q function for reinforcement learning and then perform rollouts. This approach also requires fully observed demonstrations.

8.2.4. Inverse reinforcement learning. Alternatively, we can also use the demonstrations to infer a new reward function. We use inverse reinforcement learning to infer a smoother quadratic reward function that explains the demonstrator's behavior. We infer this quadratic reward function using MaxEnt-IRL. We consider using both estimated dynamics and ground truth dynamics for this baseline. When the dynamics are estimated, this approach requires fully observed demonstrations. After inferring the

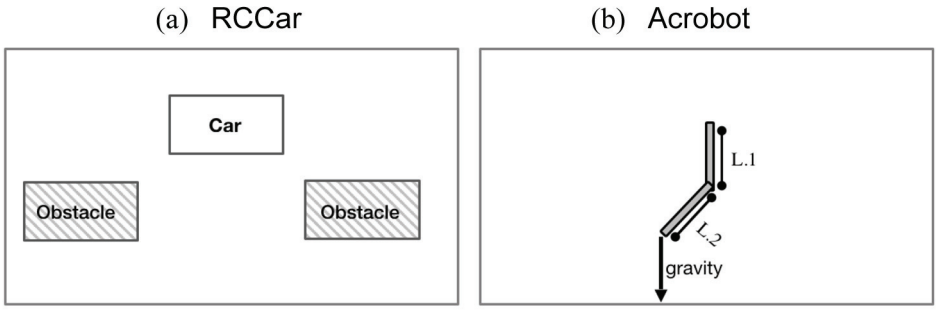

Fig. 1. (a) Simulated control task with a car with noisy nonholonomic dynamics. The car (A_1) is controlled by accelerating and turning in discrete increments. The task is to park the car between two obstacles. (b) The Acrobot domain consists of a two-link arm (L.1, L.2) with gravity and torque constraints. The task it to swing the arm above the horizon.

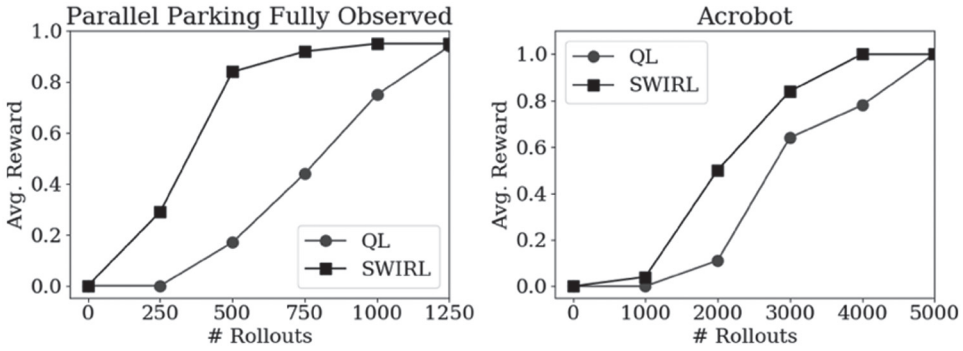

Fig. 2. Learning curves for Q-learning and SWIRL on both simulated tasks. Parallel parking: for a fixed number of demonstrations (five), we vary the number of rollouts and measure the average reward at each rollout. Acrobot: for a fixed number of demonstrations (15), we vary the number of rollouts and measure the average reward at each rollout.
QL: Q-learning; SWIRL, sequential windowed inverse reinforcement learning.

reward, the task is solved using reinforcement learning with respect to the quadratic reward function.

9. Simulated experiments

We constructed a parallel parking scenario for a robot car with nonholonomic dynamics and two obstacles (Figure 1(a)). We also experimented on the standard acrobot domain (Figure 1(b)).

For the car domain, the car can accelerate or decelerate in discrete increments of ± 0.1 m/s^2 (the car can reverse), and change its heading in increments of 5°. The car's velocity and heading θ are inputs to a bicycle steering model that computes the next state. The car observes its x position, y position, orientation, and speed in a global coordinate frame. The car's dynamics are noisy and with probability 0.1 will randomly add or subtract 2.5° to the steering angle. If the car parks between the obstacles, i.e., the speed is 0 m/s within a 15° tolerance and a positional tolerance of 1m, the task is a success and the car receives a reward of 1. The obstacles are 5 m apart (2.5 car lengths). If the car collides with one of the obstacles or does not park in 200 time-steps, the episode ends, with a reward of 0.

The acrobot domain consists of an under-actuated two-link pendulum with gravity and with torque controls on the joint. There are four discrete actions that correspond to clockwise and counterclockwise torques on each of the links. The robot observes the angle θ_1, θ_2 and angular velocity ω_1, ω_2 at each of the links. The dynamics are noisy; a small amount of random noise is added to each torque applied to the pendulum. The robot has 1000 time-steps to raise the arm above the horizontal ($y = 1$ in Figure 1b). If the task is successful, the robot receives a reward of 1. The expected reward is equivalent to the probability that the current policy will successfully raise the arm above the horizontal.

9.1. Pure exploration versus SWIRL

In the first set of experiments, we compare the learning efficiency of pure exploration with SWIRL (Figure 2). The baseline Q-learning approach is very slow because it relies on random exploration to achieve the goal at least once before it can start estimating the values of states and actions. We fix the number of initial demonstrations provided to SWIRL. We apply the segmentation and reward

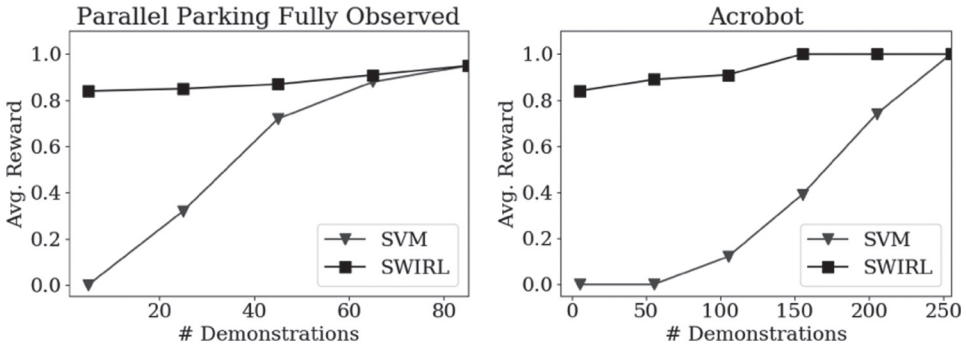

Fig. 3. Demonstration curves for imitation learning (SVM) and SWIRL on both simulated tasks. Parallel parking: we fix the number of rollouts to 500 and vary the number of demonstration trajectories that each approach observes. Acrobot: for a fixed number of rollouts (3000), we vary the number of demonstration trajectories given to each technique.
SVM: support vector machine; SWIRL, sequential windowed inverse reinforcement learning.

learning algorithms and construct a proxy task. In both domains, SWIRL significantly accelerates learning and converges to a successful policy with significantly fewer demonstrations. We find that this improvement is more substantial in the parallel parking domain. This is probably because the task more naturally partitions into discrete subtasks. In the appendixes, we visualize the segments discovered by the algorithm.

9.2. Pure demonstration versus SWIRL

Next, we evaluate SWIRL against a behavioral cloning approach (Figure 3). We collect the initial set of demonstrations and directly learn a policy using a support vector machine (SVM). For the parallel parking task, we use a linear SVM. For the acrobot task, we use a kernel SVM with a radial basis function kernel. We fix the number of autonomous rollouts that SWIRL can observe (500 for the parallel parking task and 3000 for the acrobot task). Note that the SVM technique requires observation of the actions in the demonstration trajectories, which might not be possible in all applications. The SVM approach does have the advantage that it does not require any further exploration. However, SWIRL and the pure demonstration approach are not mutually exclusive. As we show in our physical experiments, we can initialize Q-learning with a behavioral cloning policy. The combination of the two approaches allows us to take advantage of a small number of demonstrations and learn to refine the initial policy through exploration.

The SVM approach requires more than 10 times as many demonstrations to be competitive. In particular, there is an issue with exhaustively demonstrating all the scenarios that a robot might encounter. Learning from autonomous trials in addition to the initial demonstrations can augment the data without burdening the supervisor. Perhaps surprisingly, the initial dataset of demonstrations can be quite small. On both tasks, with only five demonstrations, SWIRL is within 20% of its maximum reward. Representing a policy is often more complex than representing a reward function that guides the agent to valuable states. Learning this structure requires less data than learning a full policy. This suggests that SWIRL can exploit a very small number of expert demonstrations to dramatically reduce the number of rollouts needed to learn a successful policy.

9.3. SWIRL versus other hybrid approaches

Finally, we compare SWIRL with two other hybrid demonstration–exploration approaches (Figure 4). The goal of these experiments is to show that the sequential structure learned in SWIRL is a strong prior. As in the previous experiments, it is important to note that SWIRL only requires a state trajectory as a demonstration and does not need to observe the actions taken by the expert demonstrator explicitly.

Initializing the Q function with the demonstrations, did not yield a significant improvement over random initialization. This is because one rarely observes failures in expert demonstration, and if the Q-learning algorithm does not observe poor decisions it will not be able to avoid them in the future. We also applied an inverse reinforcement learning approach using estimated dynamics. This approach is substantially better than the basic Q-learning algorithm in the parallel parking domain. This is probably because it smooths out the sparse reward to fit a quadratic function. This does serve to guide the robot to the goal states to some extent. Finally, SWIRL is the most sample-efficient algorithm. This is because the sequential quadratic rewards learned align better with the true value functions in both tasks. This structure can be learned from a small number of demonstrations.

9.4. The benefit of models

Next, we consider the benefits of using inverse reinforcement learning with the ground truth dynamic models instead of those estimated from data (Figure 5). One

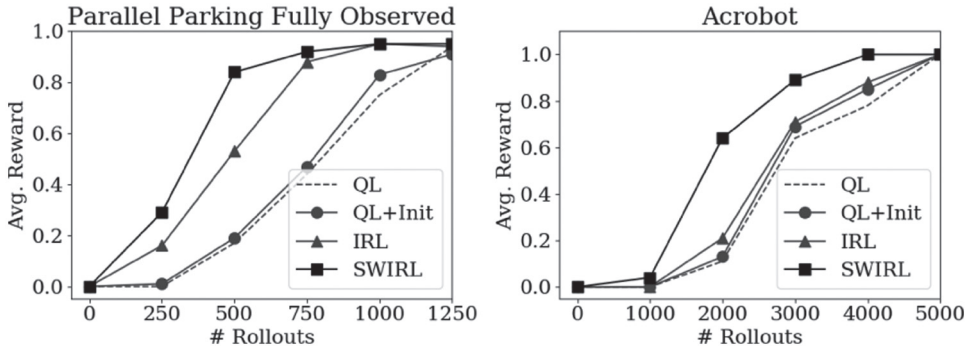

Fig. 4. Comparison of hybrid approaches. Parallel parking: for a fixed number of demonstrations (five), we vary the number of rollouts and measure the average reward at each rollout. Acrobot: for a fixed number of demonstrations (15), we vary the number of rollouts and measure the average reward at each rollout.
Init: initialization; IRL: inverse reinforcement learning; QL, Q-learning; SWIRL, sequential windowed inverse reinforcement learning.

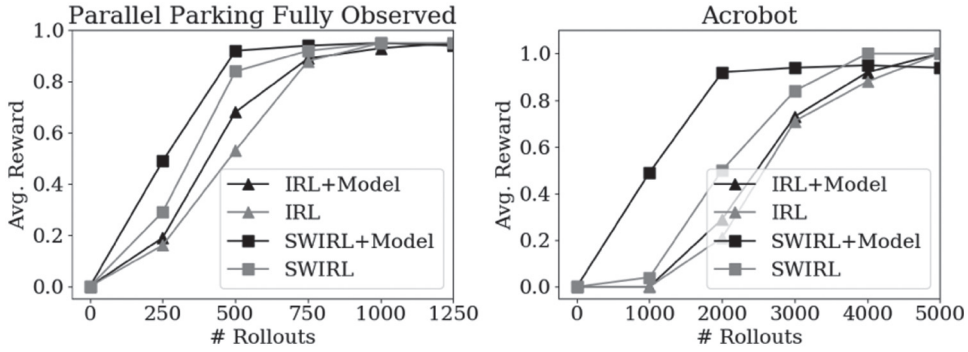

Fig. 5. Parallel parking: for a fixed number of demonstrations (five), we vary the number of rollouts and measure the average reward at each rollout. Acrobot: for a fixed number of demonstrations (15), we vary the number of rollouts and measure the average reward at each rollout.
IRL, inverse reinforcement learning; SWIRL, sequential windowed inverse reinforcement learning.

scenario where this problem setting is useful is when the demonstration dynamics are known but differ from the execution dynamics. Most common inverse reinforcement learning frameworks, such as maximum entropy inverse reinforcement learning, assume access to the dynamic model. In the previous experiments, these models were estimated from data; here, we show the benefit of providing the true models to the algorithms. Both inverse reinforcement learning and SWIRL improve their sample efficiency significantly when ground truth models are given. This experiment illustrates that the principles behind SWIRL are compatible with model-based methodologies.

9.5. Different segmentation methods

In our general framework, SWIRL is compatible with any heuristic to segment the initial demonstration trajectories. This heuristic serves to oversegment; the unsupervised learning model builds a model for sequential rewards from this heuristic. The previous experiments use an approach based on Gaussian mixture models as a segmentation heuristic; this experiment evaluates the same domains with other heuristics. In particular, we consider two other models: segmentation based on changes in the direction of velocity and segmentation based on linear dynamic regimes. Figure 6 illustrates the results. While there are differences between the performance of different heuristics, we found that the approach based on Gaussian mixture models was the most reliable across the domains.

9.6. Transfer

We constructed two transfer scenarios to evaluate whether the structure learned overfits the initial demonstrations. In essence, this is an evaluation of how well the approaches handle transfer if the dynamics change between demonstration and execution. We collected $N = 100$ demonstrations of the original task and then used the learned rewards or policies on a perturbed task. For the parallel parking task, we modified the execution environment, such that the dynamics would be coupled in a way that turning right would cause the car to accelerate forward by 0.05 m/s^2. In the perturbed task, the car must learn to adjust to this acceleration during the reversing phase. In the new domain, each

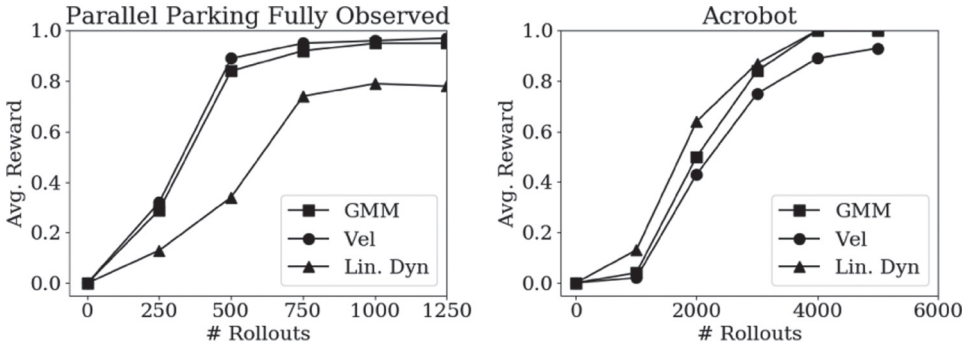

Fig. 6. We compare different transition indicator heuristics with SWIRL. Parallel parking: for a fixed number of demonstrations (five), we vary the number of rollouts and measure the average reward at each rollout. Acrobot: for a fixed number of demonstrations (15), we vary the number of rollouts and measure the average reward at each rollout.
GMM: Gaussian mixture model; Lin. Dyn.: linear dynamics; Vel.: velocity.

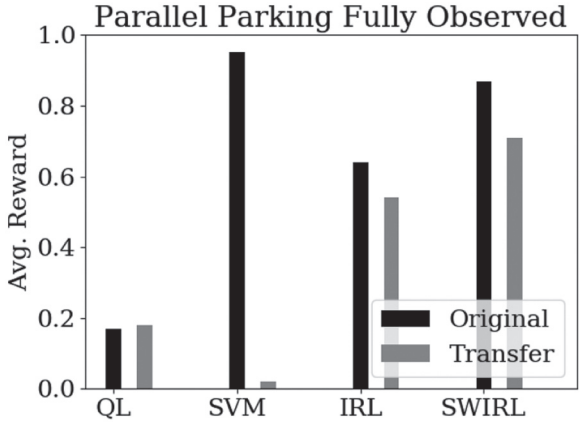

Fig. 7. For 500 rollouts and 100 demonstrations, we measure the robustness of the approaches to changes in the execution dynamics. While the SVM is 95% successful on the original domain, its success does not transfer to the perturbed setting. SWIRL learns rewards and segments that transfer to the new dynamics since they are state-space goals.
IRL: maximum entropy inverse reinforcement learning with estimated dynamics; QL: Q-learning; SVM: a baseline of behavioral cloning with a support vector machine policy representation; SWIRL: the model-free version of sequential windowed inverse reinforcement learning.

approach is allowed 500 rollouts. We report the results in Figure 7.

The success rate of the policy learned with Q-learning is more or less constant between the two domains. This is because Q-learning does not use any information from the original domain. The SVM behavioral cloning policy undergoes a drastic change. On the original domain, it achieves a 95% success rate (with 100 demonstrations); however, on the perturbed domain, it is never successful. This is because the SVM learned a policy that causes it to crash into one of the obstacles in the perturbed environment.

The inverse reinforcement learning techniques are more robust during the transfer. This is because the rewards learned are quadratic functions of the state and do not encode anything specific about the dynamics. Similarly, in SWIRL, the rewards and transition regions are invariant to the dynamics in this transfer problem. The model-free version of SWIRL reports a larger drop of 16%. This is because the model-free version is not a true inverse reinforcement learning algorithm and may encode some aspects of the dynamics in the learned reward function.

Coincidentally, this experiment also shows us how to construct a failure mode for SWIRL. If the perturbation in the task is such that it "invalidates" a transition region, e.g., a new obstacle, then SWIRL may not be able to learn to complete the task. However, the transition regions give us a formalism for detecting such problems during learning, as we can keep track of which regions are possible to reach.

For the acrobot task, as in the parallel parking scenario, we evaluate how the different approaches handle transfer if the dynamics change between demonstration and execution. With $N = 250$ demonstrations, we learn the rewards, policies, and segments on the standard pendulum; then during learning we vary the size of the second link in the pendulum. We plot the success rate (after a fixed 3000 rollouts) as a function of the increasing link size in Figure 8.

As the link size increases, even the baseline Q-learning becomes less successful. This is because the system becomes more unstable and it is harder to learn a policy. The behavioral cloning SVM policy immediately fails as the link size is increased. Inverse reinforcement learning is more robust but does not offer much of an advantage in this problem. SWIRL is robust until the change in the link size becomes large. This is because, for the larger link size, SWIRL might require different segments (or one of the learned segments is unreachable).

9.7. Sensitivity

Next, we evaluated the sensitivity of SWIRL to different initial demonstration sets (Figure 9). We sampled random initial demonstration sets and re-ran the algorithm on each of the two domains 100 times. Figure 9 is a plot of the

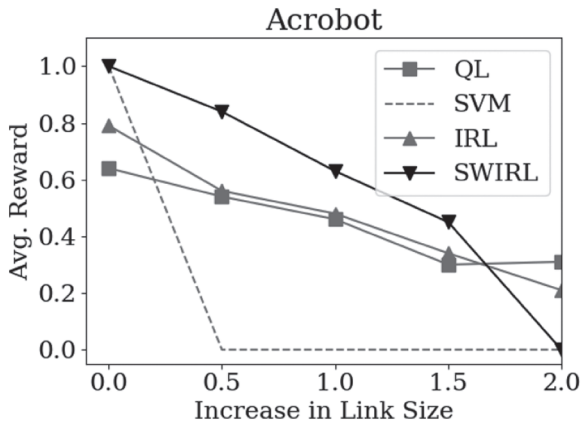

Fig. 8. For 3000 rollouts and 250 demonstrations, we measure the transfer as a function of link size. The SVM policy fails as soon the link size is changed. SWIRL is robust until the change becomes very large.
IRL: maximum entropy inverse reinforcement learning using estimated dynamics learned from the demonstrations; QL: Q-learning; SVM: a baseline of behavioral cloning with a kernel support vector machine policy representation; SWIRL: sequential windowed inverse reinforcement learning.

mean reward as a function of the number of rollouts and two standard deviations over all of the trials. We find that SWIRL is not very sensitive to the particular initial demonstration dataset. In fact, the two standard deviation error bar is smaller than the improvement in convergence in previous experiments.

9.8. Segmentation and partial observation

Next, we made the parallel parking domain more difficult, to illustrate the connection between segmentation and memory in reinforcement learning (Figure 10). We hid the velocity state from the robot, so the car only sees (x, y, θ). As before, if the car collides with one of the obstacles or does not park in 200 time-steps, the episode ends. We call this domain "parallel parking with partial observation".

This form of partial observation creates an interesting challenge. There is no longer a stationary policy that can achieve the reward. During the reversing phase of parallel parking, the car does not know that it is currently reversing. So there is ambiguity in that state; whether to pull up or reverse. We will see that segmentation can help disambiguate the action in this state.

As before, we generated five demonstrations using an RRT* motion planner (assuming deterministic dynamics) and applied each of the approaches. The techniques that model this problem using a single Markov decision process all fail to converge. The Q-learning approach achieves some nonzero rewards by chance. The learned segments in SWIRL help disambiguate dependence on history, since the segment indicator tells the car which stage of the task is currently active (pulling up or reversing) After 250,000 time-steps, the policy learned with model-based SWIRL has a 95% success rate in comparison with a $<10\%$ success rate for the baseline reinforcement learning, 0% for MaxEnt-IRL, and 0% for the SVM.

10. Physical experiments with the da Vinci surgical robot

In the next set of experiments, we evaluate SWIRL for two tasks using the da Vinci surgical robot. The da Vinci research kit is a surgical robot originally designed for teleoperation, and we consider autonomous execution of surgical subtasks. Based on a chessboard calibration, we found that the robot has a kinematic root mean square error of 3.5 mm and thus requires feedback from vision for accurate manipulation. In our robotic setup, there is an overhead endoscopic stereo camera that can be used to find visual features for learning' it is located 650 mm above the workspace. This camera is registered to the workspace with a calibration root mean square error of 2.2 mm.

10.1. Deformable sheet tensioning

In the first experiment, we consider the task of deformable sheet tensioning. The experimental setup is pictured in Figure 11. A sheet of surgical gauze is fixed at the two far

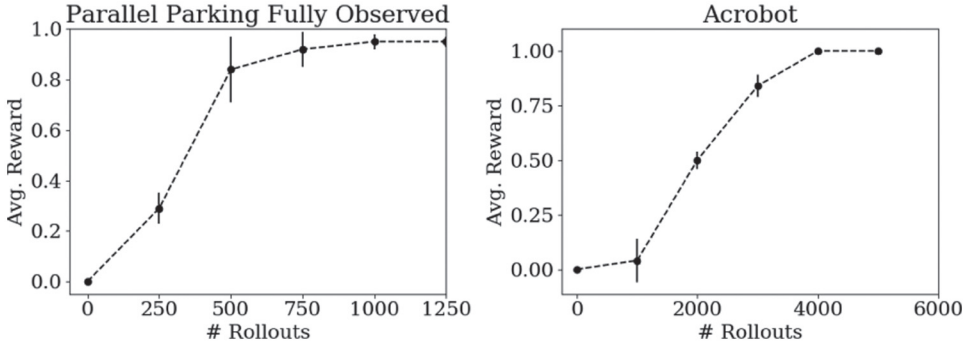

Fig. 9. Sensitivity of SWIRL. Parallel parking: we generate a random set of five demonstrations, vary the number of rollouts, and measure the average reward at each rollout. We plot the mean and standard deviation over 100 trials. Acrobot: we generate a random set of 15 demonstration, vary the number of rollouts, and measure the average reward at each rollout. We plot the mean and two standard deviations over 100 trials.

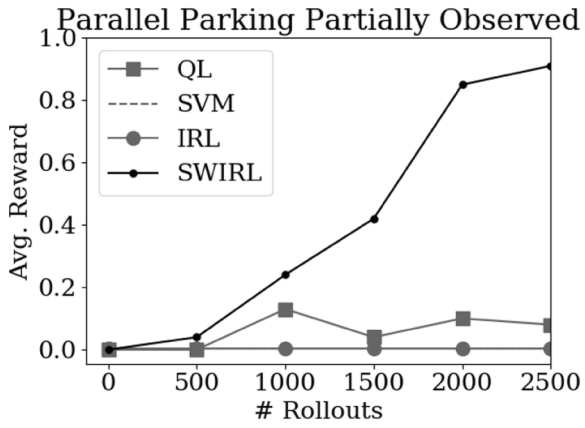

Fig. 10. We hid the velocity state from the robot, so the robot only sees (x, y, θ). For a fixed number of demonstrations (five), we vary the number of rollouts and measure the average reward at each rollout. SWIRL converges while the other approaches fail to reach a reliable success rate.
IRL: maximum entropy inverse reinforcement learning using estimated dynamics; QL: Q-learning; SVM: a baseline of behavioral cloning with a support vector machine policy representation; SWIRL: sequential windowed inverse reinforcement learning.

corners using a pair of clips. The unclipped part of the gauze is allowed to rest on soft silicone padding. The robot's task is to reach for the unclipped part, grasp it, lift the gauze, and tension the sheet to be as flat as possible. An open-loop policy typically fails in this task because it requires some feedback of whether the gauze is properly grasped, how the gauze has deformed after grasping, and visual feedback of whether the gauze is flat. The task is sequential, as some grasps pick up more or less of the material and the flattening procedure must be modified accordingly.

The state space is the six-degrees-of-freedom end-effector position of the robot, the current load on the wrist of the robot, and a visual feature measuring the flatness of the gauze. This latter is achieved using a set of fiducial markers on the gauze, which are segmented by color using a stereo camera. Then we correspond the segmented contours and estimate a z position for each marker (relative to the horizontal plane). The variance in the z position is a proxy for flatness; we include this as a feature for learning (we call this the disparity). The action space is discretized into an eight-dimensional vector ($\pm x$, $\pm y$, $\pm z$, open or close gripper) where the robot moves in 2 mm increments.

We provided 15 demonstrations through a keyboard-based tele-operation interface. The average length of the demonstrations was 48.4 actions (although we sampled observations at a higher frequency, about 10 observations for every action). From these 15 demonstrations, SWIRL identifies four segments. Figure 11 illustrates the segmentation of a representative demonstration with important states plotted over time. One of the segments corresponds to moving to the correct grasping position, one to making the grasp, one to lifting the gauze up again, and one to straightening the gauze. An interesting aspect of this task is that the segmentation requires a number of features. Figure 11 plots three signals (current load, disparity, and z position); segmenting any single signal might mean that an important feature is missed. Then, we tried to learn a policy from the rewards constructed by SWIRL. In this experiment, we initialized the policy learning phase of SWIRL using the behavioral cloning policy. We define a Q-network using a single-layer Multi-Layer Perceptron with 32 hidden units and sigmoid activation. For each of the segments, we apply behavioral cloning locally, with the same architecture as the Q-network (with an additional softmax over the output layer) to get an initial policy. We rollout 100 trials with a greedy $(\varepsilon = 0.1)$ version of these segmented policies.

The learning results of this experiment are summarized in Table 1 for different baselines. The value of the policy is a measure of the average disparity over the gauze, accumulated over the task (if the gauze is flatter for longer, the value is greater). As a baseline, we applied reinforcement learning for 100 rollouts with no other information. Using reinforcement learning did not result in successful grasping of the gauze even once. Next, we applied behavioral cloning directly. Using behavioral cloning, the gauze could be reached but not successfully grasped. Then we applied the segmentation from SWIRL and applied behavioral cloning directly to each local segment (without further refinement). Using this, the full task could be completed with a cumulative disparity score of -3516. Finally, we applied all of SWIRL and obtained the largest disparity score (-3110). For comparison, we applied SWIRL without the behavioral cloning initialization and found that success was only possible in the first two steps. This indicates that initialization is crucial in real tasks.

10.2. Surgical line cutting

In the next experiment, we evaluate generalization to different task instances. We apply SWIRL to learn to cut along a marked line in gauze, similar to the experiments of Murali et al. (2015). This is a multi-step problem, where the robot starts from a random initial state, has to move to a position that allows it to start the cut, and then cuts along the marked line. We provide the robot with five kinesthetic demonstrations by positioning the end effector and then following various marked straight lines. The state space of the robot included the end-effector position (x, y) as well as a visual feature indicating its pixel distance from the marked line *pix*. This visual feature is constructed using OpenCV thresholding for the black line. Since the gauze is planar, the robot's actions are unit steps in the $\pm x$ and $\pm y$ axes. Figure 12 illustrates the training and test scenarios.

SWIRL identifies two segments, corresponding to the positioning step and the termination. The learned reward function for the position step minimizes the x, y, pix distance to the starting point; for the cutting step, the reward function is more heavily weighted to minimize the *pix* distance. We defined task success as positioning within 1 cm

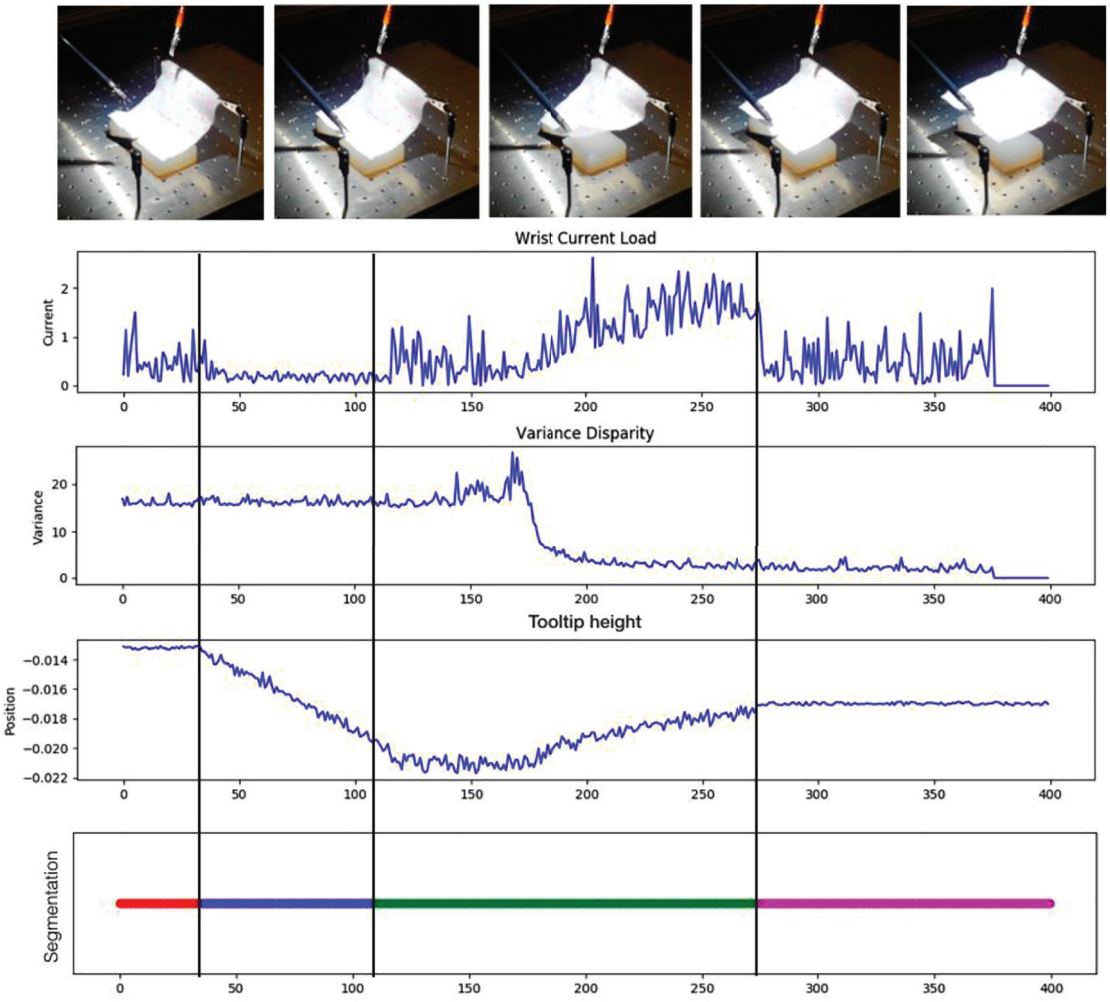

Fig. 11. Representative demonstration of deformable sheet tensioning task with relevant features plotted over time. SWIRL identifies four segments, which correspond to reaching, grasping, lifting, and tensioning. The wrist current, variance disparity (measure of smoothness of the sheet), and the tool tip height are plotted in relation to the segmentation learned by SWIRL.

Table 1. Results from the deformable sheet tensioning experiment.

Technique	No. of demonstrations	No. of rollouts	Disparity value
Pure exploration (reinforcement learning)	–	100	−8210
Pure demonstration (behavioral cloning)	15	–	−7591
Segmented demonstrations	15	–	−3516
SWIRL	15	100	

SWIRL: sequential windowed inverse reinforcement learning.

of the starting position of the line and, during the following stage, missing the line by no more than 1 cm (estimated from the pixel distance). We evaluated the model-free version of SWIRL, Q-learning, and behavioral cloning with an SVM. SWIRL was the only technique able to perform the combined task.

We evaluated the learned tracking policy to cut gauze. We ran trials on different sequences of curves and straight lines. Out of the 15 trials, 11 were successful. Two failed, owing to SWIRL errors (tracking or position was imprecise) and two failed owing to cutting errors (gauze deformed, causing the task to fail). One of the failures was on the 4.5 cm curvature line and three were on the 3.5 cm curvature line.

Next, we characterized the repeatability of the learned policy. We applied SWIRL to lines of various curvatures, spanning from straight lines to a curvature radius of 1.5 cm. Table 2 summarizes the results for lines of various curvatures. While the SVM approach did not work on the combined task, we evaluated its accuracy for each

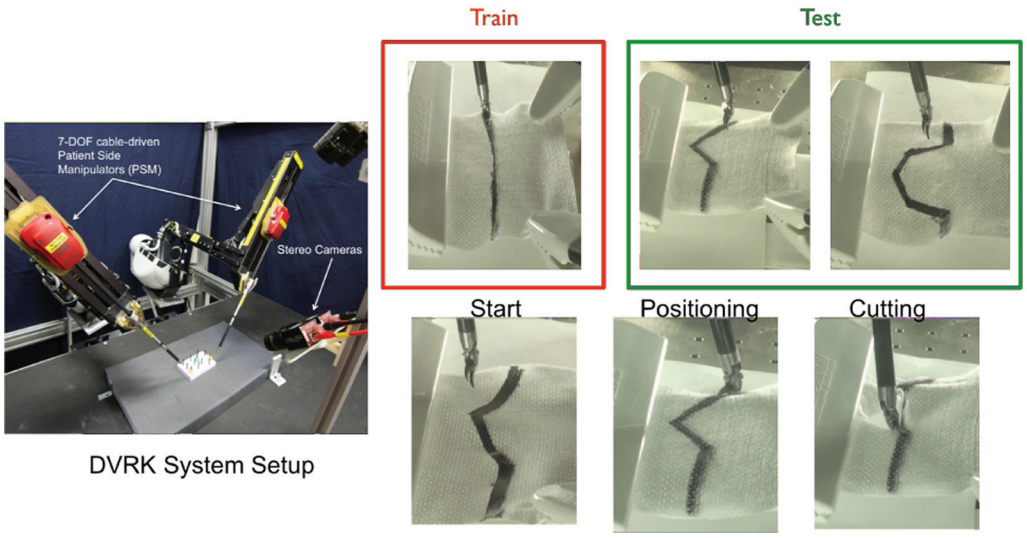

Fig. 12. We collected demonstrations on the da Vinci surgical robot kinesthetically. The task was to cut a marked line on gauze. We demonstrated the location of the line without actually cutting it. The goal is to infer that the demonstrator's reward function has two steps: position at a start position before the line, and then following the line. We applied this same reward to curved lines that started in different positions.
DVRK: da Vinci research kit.

Table 2. With five kinesthetic demonstrations of following marked straight lines on gauze, we applied SWIRL to learn to follow lines of various curvatures. After 25 episodes of exploration, we evaluated the policies on ability to position in the correct cutting location and track the line. We compare with the SVM for each individual segment. The SVM is comparably accurate on the straight line (training set) but does not generalize well to the curved lines.

Curvature radius, cm	SVM position error, cm	SVM tracking error, cm	SWIRL position error, cm	SWIRL tracking error, cm
Straight	0.46	0.23	0.42	0.21
4.0	0.43	0.59	0.45	0.33
3.5	0.51	1.21	0.56	0.38
3.0	0.86	3.03	0.66	0.57
2.5	1.43	–	0.74	0.87
2.0	–	–	0.87	1.45
1.5	–	–	1.12	2.44

SVM: support vector machine; SWIRL: sequential windowed inverse reinforcement learning.

individual step, to illustrate the benefits of SWIRL. In following straight lines, SVM was comparable to SWIRL in terms of accuracy. However, as the lines become increasingly curved, SWIRL generalizes more robustly than the SVM. A single SVM has to learn both the positioning and cutting policies. The combined policy is much more complicated than the individual policies, e.g., go to a goal and follow a line.

11. Discussion and future work

This paper explores a new algorithm, SWIRL, for segmenting tasks into shorter subtasks and assigning local reward functions. Experimental results suggest that sequential segmentation can indeed improve convergence in reinforcement learning problems with delayed rewards. Results suggest that SWIRL is robust to perturbations in initial conditions, the environment, and sensing noise. There are several limitations and avenues for future work that we would like to address.

11.1. High-dimensional state spaces

As is, SWIRL will have difficulty scaling to problems with high-dimensional state spaces, such as images. Most inverse reinforcement learning algorithms require some estimate of the dynamic model, which is difficult in general. We believe that some combination of pre-trained features and the model-free reward learning approach proposed in this paper will be a first step toward SWIRL in image space.

11.2. Avoiding reinforcement learning

Another intriguing direction is whether we can avoid the last phase of reinforcement learning. It might be possible to

design a policy learning framework that implicitly solves an inverse reinforcement learning problem. This would open a number of opportunities for incorporating segmentation, inverse reinforcement learning, and policy learning as one probabilistic model. We will also explore how the Q-learning step could be replaced with guided policy search, policy gradients, and optimal control.

11.3. More complex task structures

Another avenue for future work is the modeling of complex tasks as hierarchies of Markov decision processes, namely, tasks composed of a number of Markov decision processes that switch upon certain states and where the switching dynamics can be modeled as another Markov decision process. This is related to the options framework in hierarchical reinforcement learning; we will explore the connections between SWIRL and more complex hierarchies of behaviors.

Acknowledgments

This research was performed at the AUTOLAB at the University of California, Berkeley in affiliation with the Algorithms, Machines, and People Laboratory at the University of California, Berkeley, BAIR, and the "People and Robots" Initiative at the Center for Information Technology Research in the Interest of Society, in affiliation with the Center for Automation and Learning for Medical Robotics (Cal-MR) at the University of California, Berkeley. We thank our colleagues and the anonymous reviewers at the Workshop on the Algorithmic Foundations of Robotics who provided valuable feedback and suggestions, in particular, Pieter Abbeel, Anca Dragan, and Roy Fox.

Funding

This work was supported in part by the US National Science Foundation (NRI Award IIS-1227536), the Scalable Collaborative Human–Robot Learning (SCHooL) Project, the NSF National Robotics Initiative (grant number 1734633), Google, the Algorithms, Machines, and People Laboratory at the University of California, Berkeley, the Knut & Alice Wallenberg Foundation, a major equipment grant from Intuitive Surgical, and generous donations from Andy Chou and Susan and Deepak Lim.

References

Abbeel P and Ng AY (2004) Apprenticeship learning via inverse reinforcement learning. In: *ICML '04 proceedings of the twenty-first international conference on machine learning*, Banff, Canada, 4–8 July 2004. New York, NY: ACM.

Agrawal P, Nair AV, Abbeel P, et al. (2016) Learning to poke by poking: Experiential learning of intuitive physics. In: *NIPS'16 proceedings of the 30th international conference on neural information processing systems* (eds. DD Lee, U von Luxburg, R Garnett, et al.), Barcelona, Spain, 5–10 December 2016, pp. 5074–5082. Red Hook, NY: Curran.

Argall BD, Chernova S, Veloso M, et al. (2009) A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57(5): 469–483.

Asfour T, Azad P, Gyarfas F, et al. (2008) Imitation learning of dual-arm manipulation tasks in humanoid robots. *International Journal of Humanoid Robotics* 5(2): 183–202.

Bacon PL and Precup D (2015) Learning with options: Just deliberate and relax. In: *NIPS bounded optimality and rational metareasoning workshop*, Montreal, Canada, 12 November 2015.

Barto AG and Mahadevan S (2003) Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems* 13(1–2): 41–77.

Bertsekas DP and Tsitsiklis JN (1995) Neuro-dynamic programming: An overview. In: *Proceedings of the 34th IEEE conference on decision and control*, New Orleans, LA, USA, 13–15 December 1995, vol. 1, pp. 560–564. Piscataway, NJ: IEEE.

Botvinick MM (2008) Hierarchical models of behavior and prefrontal function. *Trends in Cognitive Sciences* 12(5): 201–208.

Botvinick MM, Niv Y and Barto AC (2009) Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition* 113(3): 262–280.

Brooks R (1986) A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation* 2(1): 14–23.

Calinon S (2014) Skills learning in robots by interaction with users and environment. In: *11th international conference on ubiquitous robots and ambient intelligence (URAI)*, Kuala Lumpur, Malaysia, 12–15 November 2014, pp. 161–162. Piscataway, NJ: IEEE.

Calinon S and Billard A (2004) Stochastic gesture production and recognition model for a humanoid robot. In: *2004 IEEE/RSJ international conference on intelligent robots and systems*, Sendai, Japan, 28 September–2 October 2004, pp. 2769–2774. Piscataway, NJ: IEEE.

Dayan P and Hinton GE (1992) Feudal reinforcement learning. In: *NIPS'92 proceedings of the 5th international conference on neural information processing systems*, Denver, CO, USA, 30 November–3 December 1992, pp. 271–278. San Francisco, CA: Morgan Kaufmann.

Dietterich TG (2000) Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* 13: 227–303.

Finn C and Levine S (2017) Deep visual foresight for planning robot motion. In: *International conference on robotics and automation (ICRA)*, Singapore, 29 May–3 June 2017, pp. 2786–2793. Piscataway, NJ: IEEE.

Finn C, Levine S and Abbeel P (2016) Guided cost learning: Deep inverse optimal control via policy optimization. In: *ICML'16 proceedings of the 33rd international conference on on machine learning* (eds. MF Balcan and KQ Weinberger), New York, NY, USA, 19–24 June 2016, pp. 49–58. Brookline, MA: Microtome Publishing.

Hengst B (2002) Discovering hierarchy in reinforcement learning with HEXQ. In: *ICML '02 proceedings of the nineteenth international conference on machine learning* (eds. C Sammut and AG Hoffmann), Sydney, Australia, 8–12 July 2002, vol. 2, pp. 243–250. San Francisco, CA: Morgan Kaufmann.

Huber M and Grupen RA (1997) A feedback control structure for on-line learning tasks. *Robotics and Autonomous Systems* 22(3–4): 303–315.

Ijspeert A, Nakanishi J and Schaal S (2002) Learning attractor landscapes for learning motor primitives. In: *NIPS'02 proceedings of the 15th international conference on neural information processing systems* (eds. S Becker, S Thrun and K Obermayer),

Vancouver, Canada, 9–14 December 2002, pp. 1523–1530. Cambridge, MA: MIT Press.

Judah K, Fern AP, Tadepalli P, et al. (2014) Imitation learning with demonstrations and shaping rewards. In: *AAAI'14 proceedings of the twenty-eighth AAAI conference on artificial intelligence*, Québec City, Canada, 27–31 July 2014, pp. 1890–1896. Palo Alto, CA: AAAI Press.

Kaelbling LP (1993) Hierarchical learning in stochastic domains: Preliminary results. In: *ICML'93 proceedings of the tenth international conference on machine learning*, Amherst, MA, USA, 27–29 July, 1993, pp. 167–173. San Francisco, CA: Morgan Kaufmann.

Karaman S and Frazzoli E (2010) Incremental sampling-based algorithms for optimal motion planning. In: *Robotics: Science and systems VI* (eds. Y Matsuoka, H Durrant-Whyte and J Neira), Zaragoza, Spain, 27–30 June 2010. Cambridge, MA: MIT Press.

Kazanzides P, Chen Z, Deguet A, et al. (2014) An open-source research kit for the da Vinci® surgical system. In: *IEEE international conference on robotics and automation (ICRA)*, Hong Kong, China, 31 May–7 June 2014, pp. 6434–6439. Piscataway, NJ: IEEE.

Kolter JZ, Abbeel P and Ng AY (2007) Hierarchical apprenticeship learning with application to quadruped locomotion. In: *NIPS'07 proceedings of the 20th international conference on neural information processing systems*, Vancouver, Canada, 3–6 December 2006, pp. 769–776. Red Hook, NY: Curran.

Konidaris G and Barto AG (2007) Building portable options: Skill transfer in reinforcement learning. In: *IJCAI'07 proceedings of the 20th international joint conference on artificial intelligence*, Hyderabad, India, 6–12 January 2007, pp. 895–900. San Francisco, CA: Morgan Kaufmann.

Krishnan S, Garg A, Liaw R, et al. (2016) SWIRL: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards. In: *Workshop on algorithmic foundations of robotics (WAFR)*, Springer Tracts in Advanced Robotics (STAR) Springer-Verlag Berlin Heidelberg San Francisco, CA, USA, 18–20 December 2016.

Krishnan S, Garg A, Patil S, et al. (2015) Transition state clustering: Unsupervised surgical trajectory segmentation for robot learning. In: *International symposium of robotics research*, Sestri Levante, Italy, 12–15 September 2015. Cham: Springer.

Kruger V, Herzog D, Baby S, et al. (2010) Learning actions from observations. *IEEE Robotics & Automation Magazine* 17(2): 30–43.

Kulis B and Jordan MI (2012) Revisiting *k*-means: New algorithms via Bayesian nonparametrics. In: *29th international conference on machine learning, ICML 2012*, Edinburgh, UK, 26 June–1 July 2012. Madison, WI: Omnipress.

Kulkarni TD, Narasimhan K, Saeedi A, et al. (2016) Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In: *NIPS'16 proceedings of the 30th international conference on neural information processing systems*, Barcelona, Spain, 5–10 December 2016, pp. 3675–3683. Red Hook, NY: Curran.

Laskey M, Lee J, Hsieh W, et al. (2017) Iterative noise injection for scalable imitation learning. *1st conference on robot learning (CoRL)*, (ed., Sergey Levine and Vincent Vanhoucke and Ken Goldberg), Mountain View, CA, USA, 13–15 November 2017, pp.143–156. PMLR.

Levine S, Pastor P, Krizhevsky A, et al. (2016) Learning hand-eye coordination for robotic grasping with large-scale data collection. In: Kulić D, Nakamura Y, Khatib O, et al. (eds.) *2016 International Symposium on Experimental Robotics. ISER 2016. Springer Proceedings in Advanced Robotics*, vol 1. Cham: Springer, pp. 173–184.

Lioutikov R, Neumann G, Maeda G, et al. (2015) Probabilistic segmentation applied to an assembly task. In: *IEEE-RAS 15th international conference on humanoid robots (humanoids)*, Seoul, South Korea, 3–5 November 2015, pp. 533–540. Piscataway, NJ: IEEE.

Manschitz S, Kober J, Gienger M, et al. (2015) Learning movement primitive attractor goals and sequential skills from kinesthetic demonstrations. *Robotics and Autonomous Systems* 74A: 97–107.

Mika S, Schölkopf B, Smola AJ, et al. (1998) Kernel PCA and denoising in feature spaces. In: *NIPS'98 proceedings of the 11th international conference on neural information processing systems*, Denver, CO, 1–3 December 1998, pp. 536–542. Cambridge, MA: MIT Press.

Mnih V, Kavukcuoglu K, Silver D, et al. (2015) Human-level control through deep reinforcement learning. *Nature* 518(7540): 529–533.

Moeslund TB and Granum E (2001) A survey of computer vision-based human motion capture. *Computer Vision and Image Understanding* 81(3): 231–268.

Morasso P (1983) Three dimensional arm trajectories. *Biological Cybernetics* 48(3): 187–194.

Murali A, Garg A, Krishnan S, et al. (2016) TSC-DL: Unsupervised trajectory segmentation of multi-modal surgical demonstrations with deep learning. In: *IEEE international conference on robotics and automation (ICRA)*, Stockholm, Sweden, 16–21 May 2016, pp. 4150–4157. Piscataway, NJ: IEEE.

Murali A, Sen S, Kehoe B, et al. (2015) Learning by observation for surgical subtasks: Multilateral cutting of 3D viscoelastic and 2D orthotropic tissue phantoms. In: *IEEE international conference on robotics and automation, ICRA*, Seattle, WA, USA, 26–30 May, 2015, pp. 1202–1209. Piscataway, NJ: IEEE.

Ng AY and Russell S (2000) Algorithms for inverse reinforcement learning. In: *ICML '00 proceedings of the seventeenth international conference on machine learning* (ed. P Langley), Stanford, CA, USA, 29 June–2 July 2000, pp. 663–670. San Francisco, CA: Morgan Kaufmann.

Ng AY, Harada D and Russell SJ (1999) Policy invariance under reward transformations: Theory and application to reward shaping. In: *ICML '99 proceedings of the sixteenth international conference on machine learning* (eds. I Bratko and S Dzeroski), Bled, Slovenia, 27–30 June 1999, pp. 278–287. San Francisco, CA: Morgan Kaufmann.

Niekum S, Osentoski S, Konidaris G, et al. (2012) Learning and generalization of complex tasks from unstructured demonstrations. In: *2012 IEEE/RSJ international conference on intelligent robots and systems, IROS*, Vilamoura, Portugal, 7–12 October 2012, pp. 5239–5246. Piscataway, NJ: IEEE.

Osa T, Pajarinen J, Neumann G, et al. (2018) An algorithmic perspective on imitation learning. *Foundations and Trends in Robotics* 7(1–2): 1–179.

Parr R and Russell SJ (1997) Reinforcement learning with hierarchies of machines. In: *NIPS'97 proceedings of the 10th international conference on neural information processing systems*

(eds. MI Jordan, MJ Kearns and SA Solla), Denver, CO, 1–6 December 1997, pp. 1043–1049. Cambridge, MA: MIT Press.
Pastor P, Hoffmann H, Asfour T, et al. (2009) Learning and generalization of motor skills by learning from demonstration. In: *International conference on robotics and automation (ICRA)*, Kobe, Japan, 12–17 May 2009, pp. 763–768. Piscataway, NJ: IEEE.
Pinto L and Gupta A (2016) Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In: *IEEE international conference on robotics and automation (ICRA)*, Stockholm, Sweden, 16–21 May 2016, pp. 3406–3413. Piscataway, NJ: IEEE.
Pinto L, Gandhi D, Han Y, et al. (2016) The curious robot: Learning visual representations via physical interactions. In: Leibe B, Matas J, Sebe N, et al. (eds.) *Computer Vision—ECCV 2016* (*Lecture Notes in Computer Science*, vol. 9906). Cham: Springer, pp. 3–18.
Ranchod P, Rosman B and Konidaris G (2015) Nonparametric Bayesian reward segmentation for skill discovery using inverse reinforcement learning. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, Hamburg, Germany, 28 September–2 October 2015, p. 471. Piscataway, NJ: IEEE.
Solway A, Diuk C, Córdova N, et al. (2014) Optimal behavioral hierarchy. *PLoS Computational Biology* 10(8): e1003779.
Stadie BC, Abbeel P and Sutskever I (2017) Third-person imitation learning. *arXiv* arXiv:1703.01703.
Sternad D and Schaal S (1999) Segmentation of endpoint trajectories does not imply segmented control. *Experimental Brain Research* 124(1): 118–136.
Sutton RS and Barto AG (1998) *Reinforcement Learning: An Introduction*, vol. 1. Cambridge, MA: MIT Press.
Sutton RS, Precup D and Singh SP (1999) Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1–2): 181–211.
Tanwani AK and Calinon S (2016) Learning robot manipulation tasks with task-parameterized semitied hidden semi-Markov model. *IEEE Robotics and Automation Letters* 1(1): 235–242.
Thrun S and Schwartz A (1993) Issues in using function approximation for reinforcement learning. In: *Proceedings of the 1993 connectionist models summer school* (eds. M Moser, P Smolensky, D Touretzky, et al.), Boulder, CO, USA, 21 June–3 July 1993. Hillsdale, NJ: Lawrence Erlbaum.
Vakanski A, Mantegh I, Irish A, et al. (2012) Trajectory learning for robot programming by demonstration using hidden Markov model and dynamic time warping. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 42(4): 1039–1052.
Viviani P and Cenzato M (1985) Segmentation and coupling in complex movements. *Journal of Experimental Psychology: Human Perception and Performance* 11(6): 828.
Whiten A, Flynn E, Brown K, et al. (2006) Imitation of hierarchical action structure by young children. *Developmental Science* 9(6): 574–582.
Zacks JM, Kurby CA, Eisenberg ML, et al. (2011) Prediction error associated with the perceptual segmentation of naturalistic events. *Journal of Cognitive Neuroscience* 23(12): 4057–4066.
Ziebart B, Dey A and Bagnell JA (2012) Probabilistic pointing target prediction via inverse optimal control. In: *IUI '12 proceedings of the 2012 ACM international conference on intelligent user interfaces*, Lisbon, Portugal, 14–17 February 2012, pp. 1–10. New York, NY: ACM.
Ziebart BD, Maas AL, Bagnell JA, et al. (2008) Maximum entropy inverse reinforcement learning. In: *AAAI'08 proceedings of the 23rd national conference on artificial intelligence* (ed. A Cohn), Chicago, IL, USA, 13–17 July 2008, vol. 3, pp. 1433–1438. Palo Alto, CA: AAAI Press.

Appendix A Parallel parking experiment

Here, we discuss some of the details of the parallel parking experiment. The car implements a dynamic bicycle model and, with probability 0.1, the car takes a random action. To implement the supervisor, a model predictive control approach was used. We applied an RRT* motion planner that re-planned to the goal state. Figure 13 illustrates the demonstrations and the segments learned by SWIRL.

Figure 14 illustrates the reward functions learned by SWIRL in each stage of the task; the matrices are plotted as a heat map. A larger value indicates that deviation in that axis is penalized more. In the first stage of the task, the cost function ensures that the robot is penalized for not matching the velocity profile of the demonstrator. The cost function is more even in the other stages.

There are also several relevant hyperparameters in this experiment.

A.1. Q-learning hyperparameters

We apply Q-learning to learn a policy for this problem with a radial basis function representation for the Q function with hyperparameters $k = 5, \sigma = 0.1$, respectively. The

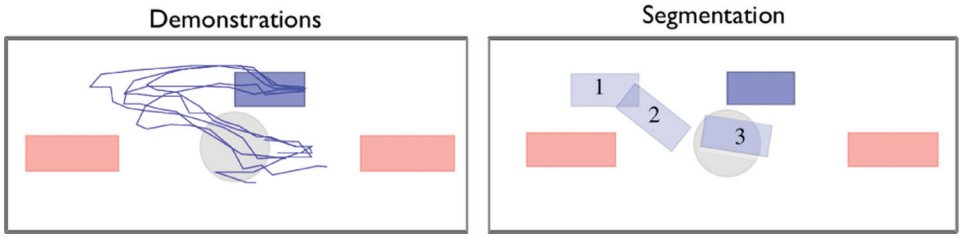

Fig. 13. Left: the five demonstration trajectories for the parallel parking task. Right: the subgoals learned by SWIRL. There are two intermediate goals corresponding to positioning the car and orienting the car correctly before reversing.
SWIRL: sequential windowed inverse reinforcement learning.

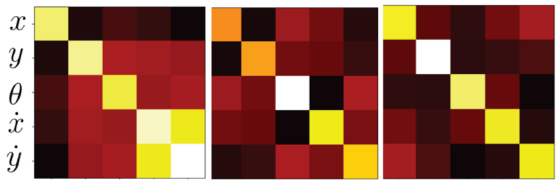

Fig. 14. Reward matrices learned in each of the segments of the parallel parking task.

radial basis function hyperparameters were tuned manually to achieve the fastest convergence in the experimental task.

A.2. Behavioral cloning hyperparameters

We use an L1 hinge-loss SVM with L2 regularization $\alpha = 5e-3$ to predict the action from the state. The hyperparameters were tuned manually using cross-validation by holding out trajectories.

Appendix B Acrobot experiment

Next, we discuss the acrobot experimental details. The acrobot implements a two-link pendulum with equal-sized links. These links are torque limited so it takes a number of swings to invert the pendulum. This system is also stochastic where the dynamics have random Gaussian noise. To implement the supervisor, a model predictive control approach was used. We applied an RRT* motion planner that re-planned to the goal state after discretizing the control inputs.

There are also several relevant hyperparameters in this experiment.

B.1. Reinforcement learning (Q-learning)

The baseline approach is to model the entire problem as a Markov decision process with the sparse delayed reward. We apply Q-learning to learn a policy for this problem with a radial basis function representation for the Q function with number of bases $k = 25$ and bandwidth $\sigma = 0.25$. The radial basis function hyperparameters were tuned manually to achieve the fastest convergence in the experimental task.

B.2. Behavioral cloning (kernel SVM)

We generated N demonstrations using the Q-learning baseline (i.e., run to convergence and sample from the learned policy). We use a radial basis function kernel SVM $\sigma = 1e-5$ with L2 regularization $\alpha = 5e-3$ to predict the action from the state. The hyperparameters were tuned manually using cross-validation by holding out trajectories.

B.3. SWIRL featurization

We apply SWIRL with a DP-GMM-based segmentation step with a kernel transformation $\sigma = 0.1$ (as described in Section 5.3).