SWIRL: A Sequential Windowed Inverse Reinforcement Learning Algorithm for Robot Tasks With Delayed Rewards

Sanjay Krishnan, Animesh Garg, Richard Liaw, Brijen Thananjeyan,
Lauren Miller, Florian T. Pokorny*, Ken Goldberg

The AUTOLAB at UC Berkeley (automation.berkeley.edu)
*CAS/CVAP, KTH Royal Institute of Technology, Stockholm, Sweden

Abstract. Delayed rewards and overfitting to training data are key challenges in robot learning. We present Sequential Windowed Inverse Reinforcement Learning (SWIRL), a three phase algorithm designed with the challenges in mind. SWIRL learns policies for sequential robot tasks using a given set of demonstrations. SWIRL first decomposes the task into sub-tasks based on Switched Linear Dynamical transitions that consistently occur across demonstrations. SWIRL then uses the inferred sequential structure to learn a sequence of local reward functions and augments the state-space with additional states to handle global dependencies. Finally, SWIRL applies policy learning to each window to compute a policy that maximizes the rewards. We compare SWIRL (demonstrations to segments to rewards to policies) with Supervised Policy Learning (SPL - demonstrations to policies) and Maximum EntropyIRL (demonstrations to rewards to policies) on standard Reinforcement Learning benchmarks: Parallel Parking with noisy dynamics, Two-Link acrobot, and a 2D GridWorld. We find that SWIRL converges to similar success rates (60%) with about 3x fewer time-steps as compared to MaxEnt-IRL, and converges to solution with 5x fewer demonstrations that SPL. At the same time SWIRL is more robust to noise than SPL, hence combining the best properties of both SPL (higher reward) and Maxent-IRL (better generalization). In physical experiments using the da Vinci surgical robot, SWIRL learns policies that generalize from linear cutting demonstrations to cutting sequences of curves where SPL fails.

1 Introduction

A primary objective of robot learning from demonstrations is to learn policies that generalize beyond the provided examples and are robust to perturbations in initial conditions, the environment, and sensing noise [1]. A popular approach is Inverse Reinforcement Learning (IRL), where the demonstrator is optimizing an unknown reward function and it is possible to infer this function from a set of demonstrations [2, 3, 4]. Once a reward is learned, given novel instances of a task, a policy can be computed by optimizing for this reward function using an approach like Reinforcement Learning (RL) [4].

In standard IRL, a task is modeled as an MDP with a stationary single reward function that maps state and action tuples to scalar values. This model is limited in the way

that it can represent *sequential tasks*, tasks where a robot must reach a sequence of intermediate state-space goals. The sequential structure can facilitate learning because the inferred reward may be *delayed* and reflect a quantity observed after all of the goals are reached, thus making it very difficult to optimize directly. Furthermore, there may not exist a single stationary policy (a time-invariant map between states and actions) that achieves all of the goals in sequence, e.g., a figure-8 trajectory in the x,y plane.

One option in such a setting is to divide the task into segments with local reward functions. In existing work on multi-step IRL, this sequential structure is defined manually [2]. We propose an approach that automatically learns sequential structure and assigns local reward functions to segments. The combined problem is nontrivial because solving k independent problems neglects the shared structure in the value function during the policy learning phase (e.g., a common failure state). However, jointly optimizing over the segmented problem inherently introduces a dependence on history, namely, any policy must complete step i before step $i+1$. Modeling an arbitrary dependence on history potentially leads to an exponential overhead of additional states. It is important to formulate a segmentation and IRL problem where this dependence can be represented efficiently.

Sequential Windowed Inverse Reinforcement Learning (SWIRL) is a three-phase algorithm for learning a policy from a set of demonstrations of a sequential task. SWIRL is based on a novel formalism for sequential tasks that represents them as a sequence of reward functions $\mathbf{R}_{seq} = [R_1, ..., R_k]$ and transition regions (subsets of the state-space) $G = [\rho_1, ..., \rho_k]$ such that R_1 is the reward function until ρ_1 is reached, after which R_2 becomes the reward and so on. SWIRL assumes that demonstrations have locally linear dynamics w.r.t a provided feature space, are locally optimal (as in IRL), and all demonstrations reach every $\rho \in G$ in the same sequence. In the first phase of the algorithm, SWIRL infers the transition regions using a kernelized variant of an algorithm proposed in our prior work [5, 6]. In the second phase, SWIRL uses the inferred transition regions to segment the set of demonstrations, and applies IRL locally to each segment to construct the sequence of reward functions $\mathbf{R}_{seq}$. In the third phase, SWIRL computes a policy using an RL algorithm (Q-Learning) over an augmented state-space that indicates the sequence of previously reached reward transition regions. We show that this augmentation has an additional space complexity independent of the state-space and linear in the number of rewards.

Our contributions are:

1. A new formalism for sequential robot tasks, where a task is modeled by an MDP with a sequence of reward functions.
2. A three-phase algorithm called SWIRL to learn policies for such tasks.
3. An extension of the Transition State Clustering algorithm that relaxes the local-linearity assumption using kernelization.
4. A novel state-space augmentation to enforce sequential dependencies using binary indicators of the previously completed segments, which can be efficiently stored and computed based on the first phase of SWIRL.
5. Simulation and Physical experiments comparing SWIRL with Supervised Learning and MaxEnt-IRL.

2 Related Work

The seminal work of Abbeel and Ng [4] explored learning from demonstrations using Inverse Reinforcement Learning. In [4], the authors used an IRL algorithm to infer the demonstrator's reward function and then an RL algorithm to optimize that reward. Our work re-visits this two-phase algorithm in the context of sequential tasks. It is well-established that RL problems often converge slowly in complex tasks when rewards are sparse and not "shaped" appropriately [7, 8]. These issues are exacerbated in sequential tasks where a sequence of goals must be reached. Related to this problem, Kolter et al. studied *Hierarchical Apprenticeship Learning* to learn bipedal locomotion [2], where the algorithm is provided with a hierarchy sub-tasks. These sub-tasks are not learned from data and assumed as given, but the algorithm infers a reward function from demonstrations. SWIRL applies to a restricted class of tasks defined by a sequence of reward functions and state-space goals.

On the other hand, there are have been some proposals in robotics to learn motion primitives from data, but they largely assume that reward functions are given (or the problem can be solved with planning-based methods). Motion primitives are example trajectories (or sub-trajectories) that bias search in planning towards paths constructed with these primitives [9, 10, 11]. Much of the initial work in motion primitives considered manually identified segments, but recently, Niekum et al. [12] proposed learning the set of primitives from demonstrations using the Beta-Process Autoregressive Hidden Markov Model (BP-AR-HMM). Calinon et al. [13] proposed the task-parametrized movement model with GMMs for action segmentation. Both Niekum and Calinon consider the motion planning setting in which analytical planning methods are used to solve a task and not RL. SWIRL considers a setting where the dynamics of the environment are noisy. Konidaris et al. studied the primitives in the RL setting [14]. However, this approach assumed that the reward function was given and not learned from demonstrations as in SWIRL. Another relevant result is from Ranchod et al. [15], who use an IRL model to define the primitives–this is in contrast to the problem of learning a policy after IRL.

3 Problem Statement and Model

3.1 Task Model

We consider the class of tasks that can be defined by a finite-horizon Markov Decision Process (MDP):

$$\mathcal{M} = \langle S, A, P(\cdot,\cdot), \mathbf{R}, T \rangle,$$

where S is the set of states (continuous or discrete), A is the set of actions (finite and discrete), $P : S \times A \mapsto Pr(S)$ is the dynamics model that maps states and actions to a probability density over subsequent states, T is the time-horizon, and $\mathbf{R}$ is a reward function that maps trajectories of length T to scalar values.

In particular, we focus on *sequential tasks* wherein the reward is accumulated sequentially. Let $\mathbf{R}_{seq} = [R_1, ..., R_k]$ be a sequence of local reward functions, where each $R_i : S \times A \mapsto \mathbb{R}$. Associated with each R_i is a sub-goal $\rho_i \subseteq S$. A task can be now be represented as a sequence of sub-goals $G = [\rho_1, ..., \rho_k]$. Each trajectory accumulates a reward R_i until it reaches the sub-goal ρ_i and continues until ρ_k is reached.

A policy is deemed *successful* when all of the $\rho_i \in G$ are reached in sequence. Further, a policy is *optimal* when it maximizes the expected cumulative reward and is successful. If we do not know $\mathbf{R}_{seq}$ and G, finding an optimal policy is challenging since it requires observing the entire trajectory and measuring the accumulated reward at the end. The key challenge in this work will be to infer $\mathbf{R}_{seq}$ and G from a given set of demonstrations. We address a particular class of sequential tasks where the dynamics of the system are locally linear, and this case G is described by the set of regions at which linear regime transitions occur.

3.2 Demonstration Model

Let D be a set of demonstrations $\{d_1, ..., d_N\}$ of a sequential task. We assume that:

1. **(Consistency)** Over all demonstrations, there exists a consistent partial order of arrival events over the set of goals $\{\rho_1, ..., \rho_k\}$.
2. **(Local Optimality)** Every demonstration is locally optimal. That is, the demonstrations are a sample from policy π_i such that for an MDP $\mathcal{M}_i$ whose reward function is R_i, π_i is the optimal stationary policy.
3. **(Local Linearity)** Let Π be the function class of allowed policies for the robot. There exists a featurization function $f : S \times A \mapsto \mathbb{R}^p$ in Π such that a trajectory $x_t \in \mathbb{R}^p$ is a linear dynamical system.

These three assumptions allow us to reduce the problem of learning G to identifying changes in dynamical motion. Consider a linear model for x_t, where A_r models the robot's dynamics, B_r models the robot's controls, and w_t is i.i.d process noise:

$$x_{t+1} = A_r x_t + B_r u_t + w_t.$$

Given this model, suppose we wanted to control the robot to a final state μ_i with a linear state-feedback controller C_i, the dynamical system that would follow is:

$$\hat{x}_{t+1} = (A_r - B_r C_i)\hat{x}_t + w_t,$$

where $\hat{x}_t = x_t - \mu_i$. If this system is stable, it will converge to $x_t = \mu_i$ as $t \to \infty$.

Now, suppose that the system has the following switching behavior: when $\|x_t - \mu_i\| \leq \varepsilon$, change the target state μ_i to μ_{i+1}. The result would be a trajectory with locally linear dynamics:

$$A_i = (A_r - B_r C_i)$$

$$x_{t+1} = A_i \mathbf{x}_t + w_t : A_i \in \{A_1, ..., A_k\}.$$

The sequence $[\mu_1, ..., \mu_k]$ and their tolerances $[\varepsilon_1, ..., \varepsilon_k]$ define the regions $[\rho_1, ..., \rho_k]$. Each ρ_i corresponds to regions where transitions occur $A_i \neq A_j$. Intuitively, a change in the reward function results in a change of policy (C_i) for a locally optimal agent. In general, there will be infinitely many feedback laws C_i that will result in a stable controller that converges to ρ_i. The choice of controller is determined by the local reward R_i. We define *generalization* as an algorithm's ability to compute a policy for a task instance $\mathcal{M}'$ with a dynamics model P', given a set of demonstrations $\mathcal{D}$ collected with respect to a task $\mathcal{M}$.

4 Sequential Windowed Inverse Reinforcement Learning

Algorithm Description Let D be a set of demonstration trajectories $\{d_1, ..., d_N\}$ of a task with a delayed reward. Given a sequence for which we require a policy, SWIRL can be described in terms of three sub-algorithms:

Inputs: Demonstrations D, Dynamics (Optional) P

1. **Sequence Learning:** Given D, SWIRL segments the task into k sub-tasks whose start and end are defined by arrival at the sub-goals $G = [\rho_1, ..., \rho_k]$.
2. **Reward Learning:** Given G and D, SWIRL associates a local reward function with the segment resulting in a sequence of rewards $\mathbf{R}_{seq}$.
3. **Policy Learning:** Given $\mathbf{R}_{seq}$ and G, SWIRL applies reinforcement learning for I iterations to learn a policy for the task π.

Outputs: Policy π

Phase I. Sequence Learning

SWIRL first infers the sub-goals $[\rho_1, ..., \rho_k]$ using an extension of our prior work on robust task segmentation [5, 6]. The overall procedure is summarized in Phase 1.

Algorithm As in [5, 6], consider a single demonstration trajectory $\mathbf{x}_t$ as a noisy observation from a dynamical system $\mathcal{T}$:

$$x_{t+1} = \mathcal{T}(x_t) + w_t,$$

where w_t describes an i.i.d noise process. We model this in a probabilistic way with a joint probability density p over a window $[x_{t-w}, ..., x_t]$ induced by the stochastic process. Since the density is non-linear, the joint distribution p can be very complex. We can model p as a GMM with k components: $p(x_{t-w}, ..., x_t) \sim GMM(k)$. This can be interpreted as defining locally linear dynamics, since conditioned on one of the mixture components, the conditional expectation $\mathbf{E}[x_t \mid x_{t-w}, ..., x_{t-1}]$ is linear, resulting in our model:

$$x_{t+1} = A_i \mathbf{x}_t + w_t : A_i \in \{A_1, ..., A_k\}.$$

Using a GMM to linearize dynamics has been widely applied [16, 17, 18]. Note that we do not need to infer the full parameters of the model $\{A_i\}$ but only *detect changes in local linearity*.

In typical GMM formulations, one must specify the number of mixture components m before hand. However, we apply results from Bayesian non-parametric statistics and jointly solve for the component locations and the number of components with a soft prior over the number of clusters using an algorithm called DP-GMM [19]. GMM-based approaches can over-segment in some problems, but we have found them to be the most reliable and easiest to tune in variety of problem settings with varying dimensionality and feature scaling.

Relaxing Local Linearity In [5, 6], we assumed that each segment has locally linear dynamics. We relax the linear dynamics assumption with a kernel embedding of the trajectories. SWIRL does not require learning the exact regimes A_i, it only needs to detect

Phase 1: Sequence Learning

Data: Demonstration $\mathcal{D}$

$\Gamma \leftarrow$ Set of all windows of size w $[\mathbf{x}_{t-w+1}, ..., \mathbf{x}_t]$

$\Theta \leftarrow$ Fit DP-GMM to Γ and find all (x_t, t) where $(x_{t+1}, t+1)$ is assigned to a different cluster.

Apply hierarchical clustering algorithm in [5] to Θ

Prune clusters that do not have one $(x_t, t) \in \Theta$ from all demonstrations.

The result of 4 is $G = [\rho_1, \rho_2, ..., \rho_m]$ where each ρ is a disjoint ellipsoidal region of the state-space and time interval.

Result: G

changes in dynamics regime. The basic idea is to apply Kernelized PCA to the features before hierarchy learning, an-oft used technique in computer vision [20]. By changing the kernel function (i.e., the similarity metric between states), we can essentially change the definition of local linearity.

Let $\kappa(x_i, x_j)$ define a kernel function over the states. For example, if κ is the radial basis function (RBF), then: $\kappa(x_i, x_j) = e^{\frac{-\|x_i - x_j\|_2^2}{2\sigma}}$. κ naturally defines a matrix M where: $M_{ij} = \kappa(x_i, x_j)$. The top p' eigenvalues define a new embedded feature vector for each ω in $\mathbb{R}^{p'}$. We can now apply the Sequence Learning in this new embedding (Phase 1).

Correspondence We model the transitions as instantaneous, and thus, in each demonstration, we have a discrete set of transition time points $\{0, ..., T\}$. We would like to be able to translate these times to state-space conditions for reward transitions $[\rho_1, ..., \rho_k]$. The set of transition states across all demonstrations induces a density over the feature-space and time. We model this density as a Gaussian Mixture Model with k mixture components $\{m_1, ..., m_k\}$ over the feature-space and time. This is learned with a two-step DP-GMM clustering described in [5], and the basic idea is to first cluster in the state-space and apply the algorithm again to cluster in time, conditioned on the state-space cluster. Therefore, each of the mixture components is a Gaussian distribution defining a region of the feature space and a time interval. Thus, the result is exactly the set of transition regions: $G = [\rho_1, \rho_2, ..., \rho_k]$.

Phase II. Reward Learning

After Phase I, each demonstration is segmented into k sub-sequences. Phase II uses the learned $[\rho_1, ..., \rho_k]$ to construct the local rewards $[R_1, ..., R_k]$ for the task. The Algorithm is summarized in Phase 2. SWIRL has two variants: when a dynamics model P is given, SWIRL applies Maximum Entropy IRL, and when the dynamics model is null (not-provided), SWIRL computes a quadratic reward around the next sub-goal.

Model-based: MaxEnt-IRL For the model-based approach, we use Maximum Entropy Inverse Reinforcement Learning (MaxEnt-IRL) [21]. The idea is to model every demonstration d_i as a sample from an optimal policy. One way to interpret this sample is that there exists a optimal trajectory d^* (in the space of all trajectories), i.e., the one that achieves the maximal reward. Each d_i that is observed is a noisy observation d^*.

In principle, one can estimate d^* from a set of demonstrations, and then use d^* to derive a reward function. However, since each d_i is a path through a possibly discrete

Phase 2: Reward Learning

Data: Demonstration $\mathcal{D}$ and sub-goals $[\rho_1, ..., \rho_k]$

1 Based on the transition states, segment each demonstration d_i into k sub-sequences where the j^{th} is denoted by $d_i[j]$.

2 If dynamics model is available, apply MaxEnt-IRL to each set of sub-sequences $1...k$.

3 If the dynamics model, is not available compute Equation 1 for each set of subsequences.

Result: $\mathbf{R}_{seq}$

state and action space, we cannot simply average them to fit a distribution. Instead, the observed data are modeled as generated with probability:

$$P(d_i|R) \propto \exp\{\sum_{t=0}^{T} R(s_t, a_t)\}$$

Paths with a higher cumulative reward are more likely.
MaxEnt-IRL uses the following linear parametrized representation:

$$R(s,a) = f(s,a)^T \theta$$

where $f(s,a)$ is the same feature vector representation used in Phase 1. The resulting form is:

$$P(d_i|R) \propto \exp\{\sum_{t=0}^{T} f(s_i, a_i)^T \theta\}$$

and MaxEnt-IRL proposes an algorithm to infer the θ that maximizes the posterior likelihood. This posterior inference procedure requires a dynamics model.

SWIRL applies MaxEnt-IRL to each segment of the task. First, applying MaxEnt-IRL to the sub-sequences of demonstrations between 0 until reaching 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 ρ_i.

Model-free: Local Quadratic Rewards When a dynamics model is not available, SWIRL uses a model-free approach for reward construction. The role of the reward function is to guide the robot to the next transition region ρ_i. SWIRL treats the robot's current state (in feature space), and its negative Euclidean distance to the ρ_i as the reward. For each $[\rho_1, ..., \rho_k]$, find the cluster centroids $[\mu_1, ..., \mu_k]$. For the segment i, we can define a reward function as follows:

$$R_i(s,a) = -\|f(s,a) - \mu_i\|_2^2$$

A problem with using Euclidean distance directly is that it uniformly penalizes disagreement with μ in all dimensions. During different stages of a task, some features will likely naturally vary more than others. To account for this, SWIRL scales this function by a matrix Q_i:

$$R_i(s,a) = (f(s,a) - \mu_i)^\intercal Q_i^{-1} (f(s,a) - \mu_i) \quad (1)$$
$$Q_i[j,l] = cov(f_j, f_l),$$

Phase 3: Policy Learning

Data: Transition States G, Reward Sequence $\mathbf{R}_{seq}$, exploration parameter ε

Initialize $Q(\binom{s}{v}, a)$ randomly
foreach $iter \in 0, ..., I$ **do**
    Draw s_0 from initial conditions
    Initialize v to be $[0, ..., 0]$
    Initialize j to be 1
    foreach $t \in 0, ..., T$ **do**
        Choose best action a based on Q or random action w.p ε.
        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

where Q 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 it gets penalized.

For example, suppose one of the features j measures the distance to a reference trajectory u_t. Further, suppose in step one of the task the demonstrator's actions are perfectly correlated with the trajectory ($Q_i[j, j]$ is low where variance is in the distance) and in step two the actions are uncorrelated with the reference trajectory ($Q_i[j, j]$ is high). Thus, Q^{-1} will respectively penalize deviation from $\mu_i[j]$ more in step one than in step two.

Phase III. Policy Learning

In Phase III, SWIRL uses the learned transitions $[\rho_1, ..., \rho_k]$ and $\mathbf{R}_{seq}$ as rewards for a Reinforcement Learning algorithm. In this section, we describe learning a policy π given rewards $\mathbf{R}_{seq}$ and an ordered sequence of transitions G.

However, this problem is non trivial since solving k independent problems neglects potential shared value structure between the 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. The key insight is that a single policy can be learned jointly over all segments over a modified problem where the state-space with additional variables that keep track of the previously achieved segments. To do so, we require an MDP model that also captures the history of the process.

MDPs with Memory RL algorithms apply to problems that are specified as MDPs. The challenge is that some sequential tasks may not be MDPs. For example, attaining a reward at ρ_i depends on knowing that the reward at goal ρ_{i-1} was attained. In general, to model this dependence on the past requires MDPs whose state-space also includes history.

Given a finite-horizon MDP $\mathcal{M}$ as defined in Section 3, we can define an MDP $\mathcal{M}_H$ as follows. Let $\mathcal{H}$ denote set of all dynamically feasible sequences of length smaller than T comprised of the elements of S. Therefore, for an agent at any time t, there is a sequence of previously visited states $H_t \in \mathcal{H}$. The MDP $\mathcal{M}_H$ is defined as:

$$\mathcal{M}_H = \langle S \times \mathcal{H}, A, P'(\cdot,\cdot), R(\cdot,\cdot), T \rangle.$$

For this MDP, P' not only defines the transitions from the current state $s \mapsto s'$, but also increments the history sequence $H_{t+1} = H_t \sqcup s$. Accordingly, the parametrized reward function R is defined over S, A, and H_{t+1}.

$\mathcal{M}_H$ allows us to address the sequentiality problem since the reward is a function of the state and the history sequence. However, without some sort of a parametrization of H_t, directly solving this MDPs with RL is impractical since it adds an overhead of $\mathcal{O}(e^T)$ states.

Policy Learning Using our sequential task definition, 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 a vector v, a k dimensional binary vector ($v \in \{0,1\}^k$) that indicates whether a transition state $i \in 0,...,k$ has been reached. This vector can be efficiently incremented when the current state $s \in \rho_{i+1}$. Then, additional complexity of representing the reward with history over $S \times \{0,1\}^k$ is only $\mathcal{O}(k)$ instead of exponential in the time horizon.

The result is an augmented state-space $\binom{s}{v}$ to account for previous progress. Over this state-space, we can apply Reinforcement Learning algorithms to iteratively converge to a successful policy for a new task instance. SWIRL applies Q-Learning with an Radial Basis Function value function representation to learn a policy π over this state-space and the reward sequence $\mathbf{R}_{seq}$. This is summarized in Algorithm 3.

5 Experiments

We evaluate SWIRL with a series of standard RL benchmarks and in a physical experiment on the da Vinci surgical robot.

5.1 Methodology

All of the experimental scenarios followed a similar pattern: (1) start with an RL problem with a delayed reward, (2) generate N demonstration trajectories with motion planning in simulated scenarios and kinethestic demonstration in the phsyical experiments, (3) apply SWIRL, (4) evaluate the performance of the policy as a function of the I iterations. For all convergence curves presented, we show the probability of task success as a function of the number of RL iterations. For convergence rate, we measure the Area Under Curve of the learning curve (i.e., cumulative expected reward over the learning epoch).

The algorithms considered in the experiments are:

1. **Q-Learning:** This applies a Q-Learning algorithm with the same hyper-parameter setting as SWIRL.
2. **Pure MaxEnt-IRL:** Given N demonstrations this learns a reward using MaxEnt-IRL and no hierarchical structure. Then, it applies Q-Learning with the same hyper-parameter setting as SWIRL until convergence. (Only Phase II and III)
3. **SVM:** Given N demonstrations this learns a policy using a multi-class SVM classifier. There is no further learning after training. (Directly to Policy)
4. **SWIRL (Model-Based) and SWIRL (Model-Free)**

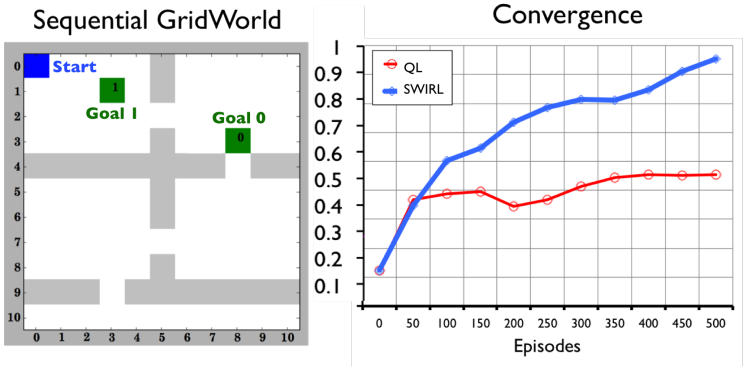

Fig. 1. GridWorld: the two goal states must be reached in sequence to receive a reward. RL algorithms find an optimal stationary policy, i.e., a time-invariant function between states and actions. In this case, Q-Learning fails to find a successful policy since a non-stationary decision is required (go right if in stage 1 of the task, go left if in stage 2). SWIRL addresses this problem by segmenting the task and applying RL with an augemented state-space.

5.2 GridWorld

We start with an example of to illustrate how delayed rewards in sequential tasks pose a challenge in RL. We modified a variant of one of the canonical RL domains in RLPy [22], GridWorld, to illustrate how SWIRL addresses problems of sequentiality (Figure 1). RL algorithms find an optimal stationary policy, i.e., a time-invariant function between states and actions. Even straight-forward sequential dependencies can introduce situations where any successful policy needs to take two different actions in the same state, depending on stage of the task.

We constructed a grid world with two goal states denoted by "0" and "1" separated by a narrow passage to illustrate the challenge of sequential dependency. The robot can only receive the reward at "1" if it has previously reached "0". In the (x, y) state-space, the robot does not learn a correct stationary policy since at some states the optimal action depends on knowing whether "0" has been reached. Thus, directly applying RL to the this problem results in an algorithm that converges to finding only one reward.

SWIRL can address this problem. We first collect 3 demonstrations of manually traced out paths that go to "0" and then "1". The robot needs to know which phase of the task it is in and during this stage which reward point is the focus. SWIRL learns three segments, and using a reward constructed with SWIRL and an augmented state-space, Q-Learning converges to find both rewards. Since GridWorld has known dynamics, we used the model-based variant of of SWIRL.

5.3 Parallel Parking

We constructed a parallel parking scenario for a robot with non-holonomic dynamics and two obstacles. The robot can control its speed ($\|\dot{x}\| + \|\dot{y}\|$) and heading ($\theta$), and observe its x position, y position, orientation, and speed in a global coordinate frame. If the robot parks between the obstacles, i.e., 0 velocity within a 15° tolerance, the task is a success and the robot receives a reward of 1. The robot's dynamics are noisy and with probability 0.1 will randomly add or subtract 5° degrees from the steering angle.

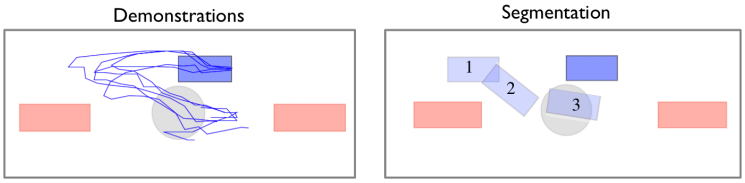

Fig. 2. This plot illustrates (a) the 5 demonstration trajectories for the parallel parking task, and (b) the sub-goals learned by SWIRL.

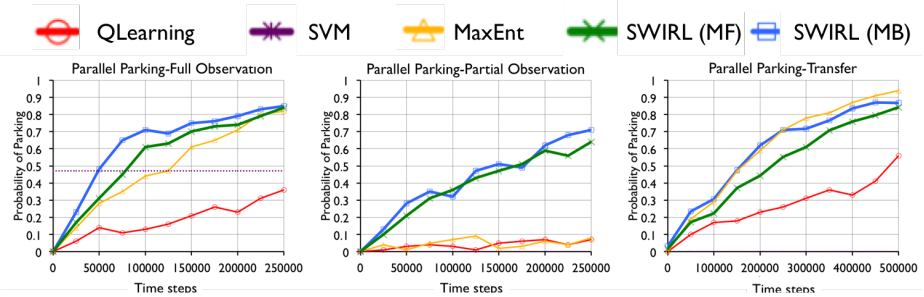

Fig. 3. Performance on a parallel parking task with noisy dynamics with full state observations (position, orientation, and velocity), partial observation (only position and orientation), and transfer (randomly permuting the action space). Success is measured in terms of the probability that the car successfully parked, and (M) denotes whether the approach used the dynamics model. In the fully observed case, both the model-based and model-free SWIRL algorithms converge faster than MaxEnt-IRL and quickly outperforms the SVM. In the partially observed case, MaxEnt-IRL, Q-Learning, and the SVM fail–while SWIRL succeeds. Both techniques also demonstrate comparable transferability to MaxEnt-IRL when the domain's dynamics are perturbed.

If the robot collides with one of the obstacle or does not park in 200 timesteps the episode ends. We call this domain Parallel Parking with Full Observation (PP-FO) (see Figure 2). Next, we made the Parallel Parking domain a little harder. We hid the velocity state from the robot, so the robot only sees (x, y, θ). As before, if the robot collides with one of the obstacle or does not park in 200 timesteps the episode ends. We call this domain Parallel Parking with Partial Observation (PP-PO).

We generated 5 demonstrations using an RRT motion planner (assuming deterministic dynamics) and applied SWIRL to learn the segments. Figure 2 illustrates the demonstrations and the learned segments. There are two intermediate goals corresponding to positioning the car and orienting the car correctly before reversing.

Performance In the first experiment (PP-FO), we use these learned segments to construct rewards in both the fully observed and partially observed problems (Figure 3). In the fully observed problem, compared to MaxEnt-IRL, the model-based SWIRL converges to a policy with a 60% success rate with about 3x fewer time-steps. The gains for the model-free version are more modest with a 50% reduction. The supervised policy learning approach achieves a success rate of 47% and the baseline RL approach achieves a success rate of 36% after 250000 time-steps.

In the partial observation problem (PP-PO), there is no longer a stationary policy that can achieve the reward. The learned segments help disambiguate dependence on history. After 250000 time-steps, the policy learned with model-based SWIRL has a

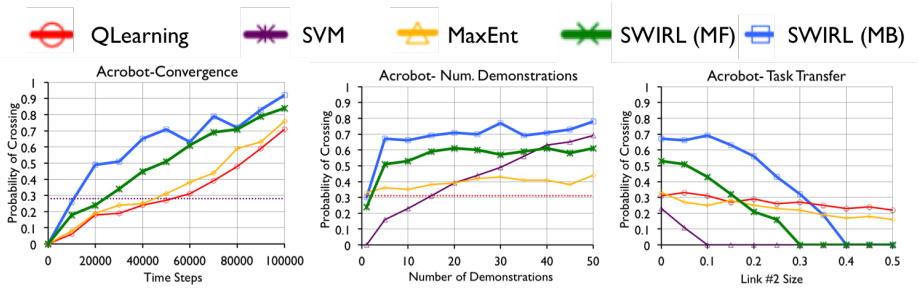

Fig. 4. Acrobot: We measured the performance of rewards constructed with SWIRL and the alternatives. We find that SWIRL (model-based and model-free) converges faster than MaxEnt-IRL, Q-Learning, and the SVM. Furthermore, SWIRL requires less demonstrations, which we measure by comparing the performance of the alternatives after a fixed 50000 time-steps and with varied input demonstrations. We also vary the task parameters by changing the size of the second link of the pendulum and find that the learned rewards are robust to this variation (as before comparing the performance of the alternatives after a fixed 50000 time steps). MaxEnt-IRL shows improved transfer performance since once the task has changed enough the segments learned during the demonstrations may not be informative and may even hurt performance if they are misleading.

70% success rate in comparison to a <10% success rate for the baseline RL, MaxEnt-IRL, and 0% for the SVM.

Next, we explore how well the constructed rewards transfer if the dynamics are changed in the fully observed setting. We expect MaxEnt-IRL to transfer well because it learns a delayed reward, which tends to encode success conditions and not task-specific details. After constructing the rewards, we randomly perturbed the system dynamics by introducing a bias towards turning left. We find that the model-based SWIRL technique transfers to this domain comparably to MaxEnt-IRL until the task is so different that the sub-goals learned with SWIRL are no longer informative. The model-free SWIRL algorithm converges more slowly–requiring 20% more time-steps to converge to the same success rate.

5.4 Acrobot

This domain consists of a two-link pendulum with gravity and with torque controls on the joint. The dynamics are noisy and there are limits on the applied torque. The robot has 1000 timesteps to raise the arm above horizontal ($y = 1$ in the images). If the task is successful and the robot receives a reward of 1. Thus, the expected reward is equivalent to the probability that the current policy will successfully raise the arm above horizontal. We generated $N = 5$ demonstrations for the Acrobot task and applied segmentation. These demonstrations were generated by training the Q-Learning baseline to convergence and then sampling from the learned policy. In Figure 4, we plot the performance of the all of the approaches. We include a comparison between a Linear Multiclass SVM and a Kernelized Multiclass SVM for the policy learning alternative. In this example, we find that applying MaxEnt-IRL does not improve the convergence rate. For this state-space, MaxEnt-IRL merely recovers the reward used in the original RL problem. On the other hand, added segments using SWIRL improve convergence rates.

We also vary the number of input demonstrations to SWIRL and find that it requires fewer demonstrations than policy learning and MaxEnt-IRL to converge to a more reliable policy. It takes about 10x more demonstrations for the supervised learning approach to reach comparable reliability. Finally, we find that SWIRL does not sacrifice much transferability. We learn the rewards on the standard pendulum, and then during learning we vary the size of the second link in the pendulum. We plot the success rate (after a fixed 50000 steps) as a function of the increase link size. SWIRL is significantly more robust that supervised policy learning to the increase in link size and has a significantly higher success rate than IRL for small perturbations in the link size.

5.5 Summary of Simulated Experiments

Table 1 summarizes the results of our experiments in terms of convergence rate and maximum attained reward on the Parallel Parking domain (with and without partial observation), Acrobot domain, and the GridWorld domains. We have additional experiments using variants of GridWorld. GridWorld-2 is a substantially harder 11x10 grid with "pits" (i.e., instant failure if reached). The Two-Bridges domain is another GridWorld based environment in which there is a short "unsafe" path between start and goal and a longer "safe" path (which is actually the optimal solution). Please refer to the arXiv report [23] for more details.

Table 1. This table summarizes the convergence rate (**AUC**) and max reward (**MAX**) attained by a Q-learning robot using the alternatives after a fixed number of iterations.

	GridWorld		GridWorld-2		Two-Bridges		PP(FO)		PP(PO)		Acrobot	
	Max	AUC	Max	AUC	Max	AUC	Max	AUC	Max	AUC	Max	AUC
Q-Learning	0.984	10.976	0.861	15.440	1.090	16.270	0.911	109.76	0.311	27.419	0.944	3.447
MaxEnt-IRL	0.987	299.556	0.861	16.956	0.759	16.270	0.950	299.556	0.444	33.128	0.920	44.111
SWIRL (MF)	1.830	322.125	1.764	14.070	**1.751**	**18.953**	**0.991**	164.127	0.934	123.115	0.906	20.935
SWIRL (MB)	**1.835**	**514.113**	**1.827**	**28.632**	1.577	17.141	0.965	**514.113**	**0.958**	**333.897**	**0.987**	**65.512**

5.6 Physical Experiments

In physical experiments, we apply SWIRL to learn to cut along a marked line in gauze similar to Murali et al. [24]. 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 cut along the marked line. We provide the robot 5 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 to 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 $\pm x, \pm y$ axes. Figure 5 illustrates the training and test scenarios.

As expected, the algorithm identifies two consistent changes in local linearity corresponding to the positioning step and the termination. The learned reward function for the position step minimizes x, y, pix distance to the starting point and 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 of the starting position of the line and during

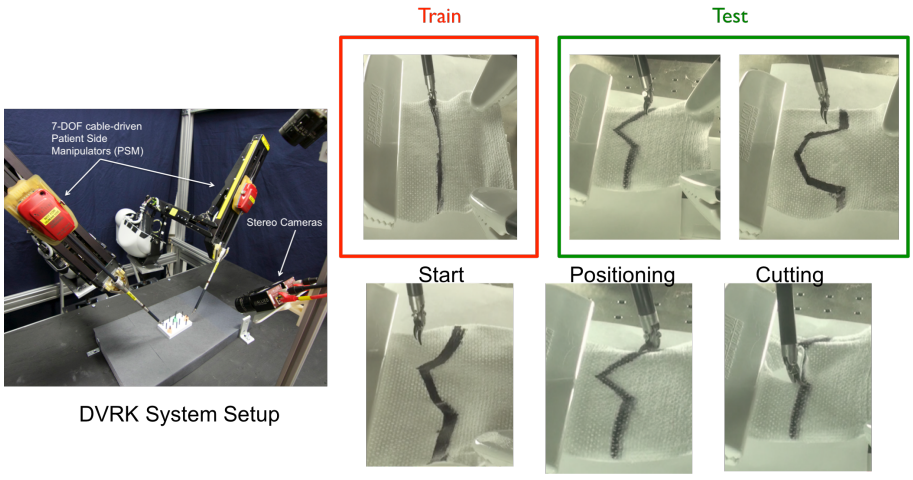

Fig. 5. We collected demonstrations on the da Vinci surgical robot kinesthetically. The task was to cut a marked line on gauze. We demonstrated the location of the line without actually cutting it. The goal is to infer that the demonstrator's reward function has two steps: position at a start position before the line, and then following the line. We applied this same reward to lines that were not straight nor started in exactly the same position.

the following stage, missing the line by no more than 1 cm (estimated from pixel distance). Since we did not have a dynamics model, we evaluated the model-free version of SWIRL, Q-Learning, and the SVM. SWIRL was the only technique able to achieve the combined task. This is because the policy for this task is non-stationary, and SWIRL is the only approach of the alternatives that can learn such a policy.

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. 2 failed due to SWIRL errors (tracking or position was imprecise) and 2 failed due to cutting errors (gauze deformed causing the task to fail). 1 of the failures was on the 4.5 cm curvature line and 3 were one the 3.5 cm curvature line.

Table 2. With 5 kinesthetic demonstrations of following marked straight lines on gauze, we applied SWIRL to learn to follow lines of various curvature. After 25 episodes of exploration, we evaluated the policies on the ability to position in the correct cutting location and track the line. We compare to SVM on each individual segment. SVM is comparably accurate on the straight line (training set) but does not generalize well to the curved lines.

Curvature Radius (cm)	SVM Pos. Error (cm)	SVM Tracking Error (cm)	SWIRL Pos. 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**

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

6 Discussion and Future Work

SWIRL is a three phase algorithm that first segments a task, learns local rewards, and learns a policy. Experimental results suggest that sequential segmentation can indeed improve convergence in RL problems with delayed rewards. Furthermore, as an algorithm based in IRL, our results suggest that SWIRL generalizes beyond the provided examples and is robust to perturbations in initial conditions, the environment, and sensing noise. This paper formalizes the interaction and composability of the three phases (sequence, reward, and policy learning), and in future work, we will explore extensions to each of the phases. We will explore how the Q-Learning step could be replaced with Guided Policy Search, Policy Gradients, or even an optimal control algorithm. In future work, we will modify the segmentation algorithm to incorporate more complex transition conditions and allow for sub-optimal demonstrations. The physical experiments in this paper present an initial demonstration of the capabilities of SWIRL, and in the future, we will explore more robotic tasks including suturing, surgical knot tying, and assembly. Another avenue for future work is modeling complex tasks as hierarchies of MDPs, namely, tasks composed of multiple MDPs that switch upon certain states with switching dynamics modeled as another MDP.

Acknowledgements: This research was performed at the AUTOLAB at UC Berkeley in affiliation with the AMP Lab, BAIR, and the CITRIS "People and Robots" (CPAR) Initiative in affiliation with UC Berkeley's Center for Automation and Learning for Medical Robotics (CalMR). The authors were supported in part by the U.S. National Science Foundation under NRI Award IIS-1227536: Multilateral Manipulation by Human-Robot Collaborative Systems, and by Google, UC Berkeley's Algorithms, Machines, and People Lab, Knut & Alice Wallenberg Foundation, and by a major equipment grant from Intuitive Surgical and by generous donations from Andy Chou and Susan and Deepak Lim. We thank our colleagues who provided helpful feedback and suggestions, in particular, Pieter Abbeel, Anca Dragan, and Roy Fox.

References

1. Argall, B.D., Chernova, S., Veloso, M., Browning, B.: A Survey of Robot Learning from Demonstration. Robotics and Autonomous Systems **57**(5) (2009) 469–483
2. Kolter, J.Z., Abbeel, P., Ng, A.Y.: Hierarchical apprenticeship learning with application to quadruped locomotion. In: Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007. (2007) 769–776
3. Coates, A., Abbeel, P., Ng, A.Y.: Learning for control from multiple demonstrations. In: Proceedings of the 25th international conference on Machine learning, ACM (2008)
4. Abbeel, P., Ng, A.Y.: Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the twenty-first international conference on Machine learning, ACM (2004) 1
5. Krishnan*, S., Garg*, A., Patil, S., Lea, C., Hager, G., Abbeel, P., Goldberg, K., (*denotes equal contribution): Transition State Clustering: Unsupervised Surgical Trajectory Segmentation For Robot Learning. In: International Symposium of Robotics Research, Springer STAR (2015)

6. Murali*, A., Garg*, A., Krishnan*, S., Pokorny, F.T., Abbeel, P., Darrell, T., Goldberg, K., (*denotes equal contribution): TSC-DL: Unsupervised Trajectory Segmentation of Multi-Modal Surgical Demonstrations with Deep Learning. In: IEEE Int. Conf. on Robotics and Automation (ICRA). (2016)
7. Ng, A.Y., Harada, D., Russell, S.J.: Policy invariance under reward transformations: Theory and application to reward shaping. In: Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Bled, Slovenia, June 27 - 30, 1999. (1999) 278–287
8. Judah, K., Fern, A.P., Tadepalli, P., Goetschalckx, R.: Imitation learning with demonstrations and shaping rewards. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada. (2014) 1890–1896
9. Ijspeert, A., Nakanishi, J., Schaal, S.: Learning attractor landscapes for learning motor primitives. In: Neural Information Processing Systems (NIPS). (2002) 1523–1530
10. Pastor, P., Hoffmann, H., Asfour, T., Schaal, S.: Learning and generalization of motor skills by learning from demonstration. In: IEEE ICRA. (2009)
11. Manschitz, S., Kober, J., Gienger, M., Peters, J.: Learning movement primitive attractor goals and sequential skills from kinesthetic demonstrations. Robotics and Autonomous Systems (2015)
12. Niekum, S., Osentoski, S., Konidaris, G., Barto, A.: Learning and generalization of complex tasks from unstructured demonstrations. In: Int. Conf. on Intelligent Robots and Systems (IROS), IEEE (2012)
13. Calinon, S.: Skills learning in robots by interaction with users and environment. In: IEEE Int. Conf. on Ubiquitous Robots and Ambient Intelligence (URAI). (2014)
14. Konidaris, G., Kuindersma, S., Grupen, R., Barto, A.: Robot Learning from Demonstration by Constructing Skill Trees. Int. Journal of Robotics Research **31**(3) (2011) 360–375
15. Ranchod, P., Rosman, B., Konidaris, G.: Nonparametric bayesian reward segmentation for skill discovery using inverse reinforcement learning. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), IEEE (2015)
16. Moldovan, T., Levine, S., Jordan, M., Abbeel, P.: Optimism-driven exploration for nonlinear systems. In: Int. Conf. on Robotics and Automation (ICRA). (2015)
17. Khansari-Zadeh, S.M., Billard, A.: Learning stable nonlinear dynamical systems with gaussian mixture models. Robotics, IEEE Transactions on **27**(5) (2011) 943–957
18. Kruger, V., Herzog, D., Baby, S., Ude, A., Kragic, D.: Learning actions from observations. Robotics & Automation Magazine, IEEE **17**(2) (2010) 30–43
19. Kulis, B., Jordan, M.I.: Revisiting k-means: New algorithms via bayesian nonparametrics. arXiv preprint arXiv:1111.0352 (2011)
20. Mika, S., Schölkopf, B., Smola, A.J., Müller, K., Scholz, M., Rätsch, G.: Kernel PCA and de-noising in feature spaces. In: Advances in Neural Information Processing Systems 11, [NIPS Conference, Denver, Colorado, USA, November 30 - December 5, 1998]. (1998) 536–542
21. Ziebart, B.D., Maas, A.L., Bagnell, J.A., Dey, A.K.: Maximum entropy inverse reinforcement learning. In: AAAI Conference on Artificial Intelligence, AAAI. (2008)
22. Geramifard, A., Klein, R.H., Dann, C., Dabney, W., How, J.P.: RLPy: The Reinforcement Learning Library for Education and Research. http://acl.mit.edu/RLPy (2013)
23. Krishnan, S., Garg, A., Liaw, R., Miller, L., Pokorny, F.T., Goldberg, K.: Hirl: Hierarchical inverse reinforcement learning for long-horizon tasks with delayed rewards. arXiv preprint arXiv:1604.06508 (2016)
24. Murali*, A., Sen*, S., Kehoe, B., Garg, A., McFarland, S., Patil, S., Boyd, W., Lim, S., Abbeel, P., Goldberg, K., (*denotes equal contribution): Learning by Observation for Surgical Subtasks: Multilateral Cutting of 3D Viscoelastic and 2D Orthotropic Tissue Phantoms. In: IEEE Int. Conf. on Robotics and Automation (ICRA). (2015)