Bayesian Nonparametric Reward Learning from Demonstration

Bernard Michini, Thomas J. Walsh, Ali-akbar Agha-mohammadi, and Jonathan P. How

***Abstract*—Learning from demonstration provides an attractive solution to the problem of teaching autonomous systems how to perform complex tasks. Reward learning from demonstration is a promising method of inferring a rich and transferable representation of the demonstrator's intents, but current algorithms suffer from intractability and inefficiency in large domains due to the assumption that the demonstrator is maximizing a *single* reward function throughout the whole task. This paper takes a different perspective by assuming that the reward function behind an unsegmented demonstration is actually composed of several distinct subtasks chained together. Leveraging this assumption, a Bayesian nonparametric reward learning framework is presented that infers multiple subgoals and reward functions within a single, unsegmented demonstration. The new framework is developed for discrete state spaces and also general continuous demonstration domains using Gaussian process reward representations. The algorithm is shown to have both performance and computational advantages over existing inverse reinforcement learning methods. Experimental results are given in both cases, demonstrating the ability to learn challenging maneuvers from demonstration on a quadrotor and a remote-controlled car.**

***Index Terms*—Reward learning, demonstration, inverse reinforcement learning**

I. Introduction

As technology continues to play a larger role in society, humans interact with autonomous systems on a daily basis. It is reasonable to assume that non-experts will increasingly interact with robotic systems and will have an idea of how the system *should* act. For the most part, however, autonomous control algorithms are currently developed and implemented by technical experts such as roboticists and computer programmers. Modifying the behavior of these algorithms is mostly beyond the capabilities of the end-user.

Learning from demonstration provides an attractive solution to this problem for several reasons [1]. The demonstrator is typically not required to have expert knowledge of the domain dynamics. This opens autonomous algorithm development to non-robotics-experts and also reduces performance brittleness from model simplifications. Also, demonstration is already an intuitive means of communication for humans, as we use demonstration to teach others in everyday life. Finally, demonstrations can be used to focus the learning process on useful areas of the state space [2], as well as provably expand the class of learnable functions [3]. There have been a wide variety successful applications that highlight the utility and potential of learning from demonstration. Many focus on teaching basic motor skills to robotic systems, such as object grasping [4], walking [5], and quadraped locomotion [6, 7]. More advanced motor tasks have also been learned [8–10]. While the previous examples are focused mainly on robotics, there are several instances of learning from demonstration in more complex, high-level tasks such as autonomous driving [11, 12], obstacle avoidance and navigation [13], and unmanned acrobatic helicopter flight [14].

This work is funded by the Office of Naval Research Science of Autonomy program under contract #N000140910625. Michini was with the Aerospace Controls Lab, MIT, Cambridge, MA, 02139. He is now Chief Technology Officer at Airware, San Francisco, CA (email: bmich@alum.mit.edu). Walsh, Agha, and How are with the Aerospace Controls Lab, MIT, Cambridge, MA, 02139. Emails: {twalsh,aliagha,jhow}@mit.edu

A. Reward Learning from Demonstration

Learning from demonstration methods can be distinguished by *what is learned* from the demonstration. Broadly, there are two classes: those that attempt to learn a *policy* from the demonstration, and those that attempt to learn a *task description* from the demonstration. In policy learning methods, the objective is to learn a mapping from states to actions that is consistent with the state-action pairs observed in the demonstration. In that way, the learned policy can be executed on the autonomous system to generate behavior similar to that of the demonstrator.

Policy methods are not concerned with *what* is being done in the demonstration, but rather *how* it is being done. In contrast, task learning methods use the demonstration to infer the objective that the demonstrator is trying to achieve. A common way of specifying such an objective is to define an associated *reward function*, a mapping from states to a scalar reward value. The task can then be more concretely defined as reaching states that maximize accumulated reward. This paper focuses primarily on the problem of *reward learning from demonstration* via the discovery of *subgoals*.

Reward learning is challenging for several fundamental reasons. First, learning a reward function from demonstration necessitates a *behavioral model* of the demonstrator that predicts what actions would be taken given some reward function (or objective). The actions predicted by the model are compared to the demonstration as a means of inferring the reward function of the demonstrator. The demonstrator model is typically difficult to obtain in that it requires solving for a policy that maximizes a candidate reward function. Methods that require only sample access to the demonstrator alleviate this problem. Second, the demonstration typically admits many possible corresponding reward functions, i.e., there is no unique reward function that explains a given set of observed actions. Finally, the demonstration itself can be inconsistent and the demonstrator imperfect. Thus, it cannot be assumed that the actions in the demonstration optimize accumulated reward, only that they *attempt* to.

Despite these difficulties, reward learning has several perceived advantages over policy learning. A policy, due to its nature as a direct mapping from states to actions, becomes invalid if the state transition model changes (actions may have different consequences than they did in the original demonstration). Also, a learned policy mapping must be defined for each state to be encountered, relying on a large demonstration set or additional generalization methods. A learned reward function, however, can be used to *solve for a policy* given knowledge of the state transition model, making it invariant to changes in domain dynamics and generalizable to new states. Thus, a reward function is a succinct and transferable description of the task being demonstrated and still provides a policy which generates behavior similar to that of the demonstrator.

This paper develops reward learning techniques that are scalable to large, real-world, continuous demonstration domains while retaining computational tractability. Achieving both of these goals generally requires assumptions to be made about the structure of the reward function (e.g., [11, 15]). In this work, the following assumption (a more formal instantiation of which is presented in Definition 1) is made about the reward function:

Assumption 1. *The reward function underlying the demonstrator's behavior is assumed to be composed of several subtask reward functions based on subgoals, with the behaviors in each subtask chained together sequentially.*

There is a long history of using such subgoal-based rewards, and the technique has been successful at modeling a diverse set of tasks [16–19]. Furthermore, this focus on subtasks within a larger demonstration provides a practical alternative to many previous reward learning methods that assume a single monolithic reward function is responsible for the demonstration. Because of this assumption, the core component for reward learning in this paper is the learning of subgoals from demonstrations. However, the framework presented here requires these identified subgoals be transformed into a reward function to assess their fit to the demonstration and ultimately execute learned behavior.

Based on this assumption, two new reward-learning methods are presented that utilize Bayesian nonparametric mixture models to simultaneously partition the demonstration and learn associated reward functions. While the type and complexity of reward functions learned are not fundamentally constrained by the algorithms presented, to clarify the process, the methods are developed using simple state-based and feature-based reward functions. Several key approximation methods are also developed with the aim of improving efficiency and tractability in large continuous domains. Simulation results are given which highlight key properties and advantages, and experimental results validate the new algorithms applied to challenging robotic systems.

The next subsection highlights relevant previous work in the field of learning from demonstration, which is followed by a more detailed summary of the paper contributions.

B. Related Work

Because of Assumption 1, inferring the particular subgoals leading to the demonstrated behavior is the central component of the reward learning process. However, in order to fit the observed trajectories to particular subgoals and actually execute behavior, the framework also calculates a full reward function based on the subgoals. Thus this work has features of Inverse Reinforcement Learning (IRL) algorithms that learn reward functions as well as goal inference algorithms. We describe methods from both areas below.

Of the learning from demonstration methods that learn a task description, most do so by learning a reward function that rationalizes the observed demonstrations. This is typically done with a known (at least approximately) model of the system dynamics. In the context of control theory, the problem of finding such a reward function is known as Inverse Optimal Control, originally posed by Kalman and solved in [20]. Ng and Russell cast the problem in the reinforcement learning framework [21] and termed it Inverse Reinforcement Learning (IRL), highlighting the fact that the reward function in many RL applications is not known *a priori* and must instead be learned. IRL seeks to learn the reward function which is argued in [21] to be the "most succinct, robust, and transferable definition of the task".

There have since been a number of IRL methods developed, and most make assumptions about the nature of the reward function (as is done in Assumption 1 above). For instance, many popular methods (e.g., [11, 15]) use a weighted-features representation for the unknown reward function. Specifically, the reward function is represented as

$$R(s) = w^T \phi(s) \tag{1}$$

where w is a vector of weights and ϕ is a function mapping a state s to a feature vector. In such a setting, Abbeel and Ng solve a quadratic program iteratively to find feature weights that attempt to match the expected feature counts of the resulting policy with those of the expert demonstrations [11]. A game-theoretic approach for the same problem is taken in [15], whereby a minimax search is used to minimize the difference in weighted feature expectations between the demonstrations and learned policy. In [22], IRL is generalized to multiple tasks from multiple demonstrations via formalizing it as a statistical preference elicitation. Ratliff et al. [6, 23] take a max-margin approach, finding a weight vector that explains the expert demonstrations by optimizing the margin between competing explanations. Ziebart et al. [24] match feature counts using the principle of maximum entropy to resolve ambiguities in the resulting reward function. In [25], the parameters of a generic family of parametrized rewards are found. Ramachandran and Amir [26] learn a finite vector of reward parameters using Bayesian Inverse Reinforcement Learning (BIRL). The above approaches assume a given parametrization of the reward function (such as the weighted-features representation in (1)) and learn a vector of parameters.

These IRL algorithms are similar in that they attempt to find a single reward function that explains the entirety of the observed demonstration. These approaches have been highly successful in learning to perform a number of tasks such as simulated driving [11] and adventure games [26]. However, such reward functions can have difficulties capturing natural behavior in sequential robot tasks. For instance, consider a

robot performing a round-trip to and from a point with features representing the distance to the origin and distance to the turn-around point. Unfortunately, these features are not enough to capture a linear reward function of the type described in (1), since the weight on the first feature needs to be higher on the way out, but the weight on the second feature needs to be higher on the way back. Doubling the number of features by indicating the conjunction of distance with the *stage* of the task (e.g., 0.3 meters from the origin and in the "return" stage) makes the representation feasible, but increases the complexity of reward learning for two reasons. First, as the complexity of the reward model increases (i.e., as more features are added to ϕ), so too does the number of free parameters that need to be learned. Thus, the dimension of the parameter space is larger and the search for candidate functions becomes increasingly difficult according to the curse of dimensionality. Second, the process of evaluating candidate reward functions against the demonstrated trajectories requires solving for the optimal policy given the candidate reward function, the computational cost of which typically scales poorly with domain size, even for approximate solutions [27]. Thus finding a single, complex reward function to explain the observed demonstrations requires searching over a large space of possible solutions and substantial computational effort to evaluate each candidate. Further analysis comparing the computational complexities of previous methods versus the proposed algorithm is given in Section III-G.

In an attempt to more efficiently model situations like the "round-trip" scenario, this work makes a different assumption than the linearity constraints above. Instead, it assumes (under Assumption 1 and more formal definitions to follow) that the demonstrator's reward function is composed of many reward functions and is partitioned by subgoals. As an example, the monolithic reward function in "round-trip" can be partitioned into two reward functions, with the turn-around acting as a subgoal. The *Bayesian Nonparametric Inverse Reinforcement Learning* (BNIRL) method introduced in this paper searches for such a partitioning and the corresponding reward functions (one for each partition), thereby avoiding the potential intractability of searching for a single reward function. We note that while this assumption makes it easier to learn the "round-trip" behavior, other tasks such as a continuous driving simulator [11] will be easier to represent with the linearity assumption behind Equation 1.

Several methods have been developed that also address the issue of identifying multiple subtasks within an unsegmented demonstration. Grollman et al. characterize the demonstration as a mixture of Gaussian process experts [28] and find multiple policies describing the demonstration. However, that method learns such a policy directly, without building a reward function to represents the task itself. Thus, if the system dynamics change, the learned policy may not complete the previously demonstrated task. By contrast, our BNIRL method learns such a reusable reward function. Fox et al. also use a Bayesian nonparametric framework, but cast the demonstration as a switched linear dynamic system, and infer a hidden Markov model to indicate switching between systems [29]. However, that work is focused on segmentation, not reward learning or actually enacting a learned policy, which is the core of this work. Extracting motion primitives is also a common strategy for segmenting continuous trajectories, e.g., by detecting dynamic switches [30] or by identifying switched latent force models [31]. However, neither of these methods attempt to learn multiple *reward* functions from unsegmented demonstration, which is advantageous for the reasons discussed in Section I-A. Finally, several methods exist which learn sets of attractors [32] or via-points [33, 34] from a trajectory. These are similar in that they capture subgoal-like intent, but do not use Bayesian nonparametric techniques to explicitly learn the number of subgoals. Also, in contrast to attractors and via-points, the subgoal reward representation in this work can be extended to more complex feature parametrizations.

Throughout the paper, subgoals are used as simple reward representations to explain partitioned demonstration data. The notion of defining tasks using a corresponding subgoal was proposed by Sutton et al. along with the options MDP framework [16]. Many other methods exist that learn options from a given set of trajectories. For example, in [35], diverse density across multiple solution paths are used to discover such subgoals. Several algorithms use graph-theoretic measures to partition densely-connected regions of the state space and learn subgoals at bottleneck states [36, 37]. Bottleneck states are also identified using state frequencies [38] or using a local measure of relative novelty [39]. Of these methods, all require multiple trial trajectories to learn subgoals and, furthermore, none have the ability to learn *reward* functions from demonstration. The latter is an important distinction because identifying a bottleneck state does not automatically induce a policy for actually reaching that state, while BNIRL's inference of a reward model combined with the known dynamics model does induce such a policy. BNIRL is also shown to work with small amounts of demonstration data, and in many cases with just a single trajectory.

One similar algorithm from the subgoal discovery literature is the segmentation and skill construction algorithm of Niekum et al. [18], which uses a Bayesian nonparametric clustering technique to partition trajectories and then learns Dynamic Motion Primitives (DMPs) to represent the skill used in each segment. DMPs model each of the partitions using a set of nonlinear differential equations and ultimately provide a set of weights for controllers that determine the policy for that segment. While this algorithm has many of the components of BNIRL, its assumptions and domains of application are significantly different. BNIRL can be applied in general Markov Decision Processes (MDPs), both discrete or continuous, creates partitions based on changes in the user's actions, and can learn reward functions for each partition. These BNIRL reward functions are then combined with a known dynamics model to generate a policy. By contrast, Niekum et al. do not assume a dynamics model is available and do not directly use changes in the demonstrator's action selection to perform segmentation. Instead, they assume that each skill can be represented by a DMP dynamics model, which limits their approach to continuous domains where DMPs can be applied (as opposed to the general MDP reward functions of BNIRL). Also, learning DMP descriptions of skills produces a set of

controller weights, essentially a policy for each segment, not a reward function. Thus, if the underlying dynamics of the system change, then the task will need to be relearned. This learning of DMP policies is a significant difference from reward learning using a general MDP model as performed by BNIRL.

C. Main Contributions

This paper focuses broadly on improving existing reward learning from demonstration methods and developing new methods that enable scalable reward learning for real-world robotic systems. We present a new reward learning framework called BNIRL, which uses a Bayesian nonparametric mixture model to automatically partition the data and find a set of simple reward functions corresponding to each partition. The simple rewards are interpreted intuitively as subgoals, which can be used to predict actions or analyze which states are important to the demonstrator. Convergence of the BNIRL algorithm in 0-1 loss is proven and several computational advantages of the method over existing IRL frameworks are shown, namely the search over a finite (as opposed to infinite) space of possible rewards and the ability to easily parallelize most of the BNIRL computational requirements.

Two approximations to the demonstrator likelihood function further improve computational tractability in large domains. One approach uses the Real-time Dynamic Programming (RTDP) [40] framework to approximate the optimal action-value function. RTDP effectively limits computation of the value function to necessary areas of the state space, allowing the complexity of the BNIRL reward learning method to scale with the size of the demonstration set, *not* the size of the full state space. In the second method, an existing closed-loop controller takes the place of the optimal value function. This avoids having to specify a discretization of the state or action spaces, extending the applicability of BNIRL to continuous demonstration domains when a closed-loop controller is available.

BNIRL can learn multiple reward functions from a single demonstration, but it is only generally applicable in discrete domains. The Bayesian nonparametric reward learning framework is extended to general continuous demonstration domains using Gaussian process reward representations. The resulting algorithm, termed Gaussian process subgoal reward learning (GPSRL), is the only learning from demonstration method able to learn multiple reward functions from unsegmented demonstrations in general continuous domains. GPSRL does not require discretization of the continuous state space and focuses computation efficiently around the demonstration itself. Learned subgoal rewards are cast as Markov decision process options to enable execution of the learned behaviors by the robotic system and provide a principled basis for future learning and skill refinement. A method is developed for choosing the key confidence parameter in the GPSRL likelihood function, and furthermore this method can be used to quantify the relative skill level of the demonstrator enabling comparison between multiple demonstrators.

While simulation results are given throughout, the final contribution of the paper is to provide experimental results validating the ability of the proposed methods to learn reward from demonstrations in real-world robotic domains. Quadrotor flight maneuvers are learned from a human demonstrator using only hand motions with the BNIRL algorithm. The GPSRL algorithm is experimentally applied to a robotic car domain, learning multiple difficult maneuvering skills such as drifting turns from a single unsegmented demonstration.

This work combines both previously-published and unpublished algorithms and results. The BNIRL framework and associated simulation results in Section III first appeared in [41]. The approximations methods to enable scalability of BNIRL in Section IV and the associated quadrotor helicopter experimental results in Section VI-A first appeared in [42]. This is the first refereed publication presenting the GPSRL framework in Section V and the associated remote-controlled car experiments in Section VI-B.

II. Background

This section provides a background in the mathematical concepts that this paper builds upon. Throughout the paper, boldface is used to denote vectors and subscripts are used to denote the elements of vectors (i.e., z_i is the ith element of vector $\boldsymbol{z}$).

A. Markov Decision Processes and Options

A finite-state Markov Decision Process (MDP) is a tuple (S, A, T, γ, R) where S is a set of *states*, A is a set of *actions*, $T : S \times A \times S \mapsto [0, 1]$ is the function of *transition probabilities* such that $T(s, a, s')$ is the probability of being in state s' after taking action a from state s, $R : S \mapsto \mathbb{R}$ is the *reward function*, and $\gamma \in [0, 1)$ is the *discount factor*. A *stationary policy* is a function $\pi : S \mapsto A$. Ref. [43] provides the following definitions and results:

1) The infinite-horizon expected reward for starting in state s and following policy π thereafter is given by the *value function* $V^\pi(s, R)$:

$$V^\pi(s, R) = E_\pi \left[\sum_{i=0}^{\infty} \gamma^i R(s_i) \middle| s_0 = s \right] \tag{2}$$

Assuming state-based reward (i.e., rewards that do not depend on actions, as in [26]), the value function satisfies the following Bellman equation for all $s \in S$:

$$V^\pi(s, R) = \sum_{s'} T(s, \pi(s), s') \left[R(s') + \gamma V^\pi(s') \right] \tag{3}$$

The so-called Q-function $Q^\pi(s, a, R)$ is defined as the infinite-horizon expected reward for starting in state s, taking action a, and following policy π thereafter.

2) A policy π is optimal iff, for all $s \in S$:

$$\pi(s) = \operatorname*{argmax}_{a \in A} Q^\pi(s, a, R) \tag{4}$$

An optimal policy is denoted as π^* with corresponding value function V^* and action-value function Q^*.

An *option*, o, is a temporally-extended action defined by the tuple (I_o, π_o, β_o) [16]. $I_o : S \mapsto \{0, 1\}$ is the *initiation set*, defined to be 1 where the option can be executed and 0 elsewhere. $\pi_o : S \mapsto A$ is the *option policy* for each state where the option is defined according to I_o. Finally, $\beta_o : S \mapsto$

$[0, 1]$ is the *terminating condition*, defining the probability that the option will terminate in any state for which the option is defined. Any method which creates new skills (in the form of options) must define at least I_o and β_o. The option policy π_o can be learned using standard RL methods.

B. Inverse Reinforcement Learning

Inverse reinforcement learning (IRL) [21] is the problem of inferring the reward function responsible for generating observed optimal behavior. Formally, IRL assumes a given MDP/R, defined as a MDP for which everything is specified except the state reward function $R(s)$. Observations (demonstrations) are provided as a set of state-action pairs:

$$\boldsymbol{O} = \{(s_1, a_1), (s_2, a_2), ..., (s_N, a_N)\} \tag{5}$$

where each pair $O_i = (s_i, a_i)$ indicates that the demonstrator took action a_i while in state s_i. It is assumed that states and actions are fully-observable, and problems associated with partial state/action observability are not considered. Inverse reinforcement learning algorithms attempt to find a reward function that rationalizes the observed demonstrations, i.e., find a reward function $\widehat{R}(s)$ whose corresponding optimal policy π^* matches the observations $\boldsymbol{O}$.

C. Chinese Restaurant Process Mixtures

The algorithms developed throughout the paper combine IRL with a Bayesian nonparametric model for learning multiple reward functions, namely the Chinese restaurant process mixture model. The *Chinese restaurant process* (CRP) is a sequential construction of random partitions used to define a probability distribution over the space of all possible partitions [44]. The process by which partitions are constructed follows a metaphor whereby customers enter a Chinese restaurant and must choose a table. In the analogy, tables are used to represent partitions, and the Chinese restaurant has a potentially infinite number of tables available. The construction proceeds as follows:

1) The first customer sits at the first table.
2) Customer i arrives and chooses the first unoccupied table with probability $\frac{\eta}{i-1+\eta}$, and an occupied table with probability $\frac{c}{i-1+\eta}$, where c is the number of customers already sitting at that table.

The concentration hyper-parameter η controls the probability that a customer starts a new table. Using $z_i = j$ to denote that customer i has chosen table j, C_j to denote the number of customers sitting at table j, and J_{i-1} to denote the number of tables currently occupied by the first $i-1$ customers, the assignment probability is defined as:

$$P(z_i = j | z_{1...i-1}) = \begin{cases} C_J(i-1+\eta)^{-1} & j \leq J_{i-1} \\ \eta(i-1+\eta)^{-1} & j = J_{i-1}+1 \end{cases} \tag{6}$$

This process induces a distribution over table partitions that is *exchangeable* [45], meaning that the order in which customers arrive can be permuted and any partition with the same proportions will have the same probability. A Chinese restaurant process mixture is defined using the same construct, but each table is endowed with parameters θ of a probability distribution which generates data points x_i:

1) Each table j is endowed with parameter θ_j drawn i.i.d. from a prior $P(\theta)$.
2) For each customer i that arrives:
   a) The customer sits at table j according to (6) (the assignment variable $z_i = j$).
   b) A data-point x_i is drawn i.i.d. from $P(x|\theta_j)$.

The ability of the CRP mixture to model data that are generated from a random and potentially infinite number of partitions is critical to the algorithms in this paper.

D. Gaussian Processes

A *Gaussian process* (GP) is a distribution over functions, widely used in machine learning as a nonparametric regression method for estimating continuous functions from sparse and noisy data [46]. In this paper, Gaussian processes will be used as a subgoal reward representation which can be trained with a single data point but has support over the entire state space. A training set consists of input vectors $\boldsymbol{X} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_n]$ and corresponding observations $\boldsymbol{y} = [y_1, ..., y_n]^\top$. The observations are assumed to be noisy measurements from the unknown target function f, i.e., $y_i = f(\boldsymbol{x}_i) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$ is Gaussian noise. A zero-mean Gaussian process is completely specified by a covariance function $k(\cdot, \cdot)$, called a *kernel*. Given the training data $\{\boldsymbol{X}, \boldsymbol{y}\}$ and covariance function $k(\cdot, \cdot)$, the Gaussian process induces a predictive marginal distribution for test point $\boldsymbol{x}_*$ which is Gaussian distributed so that $f(\boldsymbol{x}_*) \sim \mathcal{N}(\mu_{f_*}, \sigma_{f_*}^2)$ with mean and variance given by:

$$\mu_{f_*} = k(\boldsymbol{x}_*, \boldsymbol{X}) \left(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y} \tag{7}$$

$$\sigma_{f_*}^2 = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - k(\boldsymbol{x}_*, \boldsymbol{X}) \left(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I}\right)^{-1} k(\boldsymbol{X}, \boldsymbol{x}_*) \tag{8}$$

where $\boldsymbol{K} \in \mathbb{R}^{n \times n}$ is the Gram matrix with $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

Selecting a kernel is typically application-specific, since the function $k(\boldsymbol{x}, \boldsymbol{x}')$ is used as a measure of correlation (or distance) between states $\boldsymbol{x}$ and $\boldsymbol{x}'$. A common choice (used widely throughout the paper) is the squared exponential (SE) kernel $k_{\text{SE}}(\boldsymbol{x}, \boldsymbol{x}') = \nu^2 \exp(-.5(\boldsymbol{x}-\boldsymbol{x}')^\top \Lambda^{-1}(\boldsymbol{x}-\boldsymbol{x}'))$, where $\Lambda = \text{diag}(\lambda_1^2, ..., \lambda_{n_x}^2)$ are the characteristic length scales of each dimension of $\boldsymbol{x}$ and ν^2 describes the variability of f. Thus $\theta_{\text{SE}} = \{\nu, \lambda_1, ..., \lambda_{n_x}\}$ is the vector of hyper-parameters which must be chosen for the squared exponential kernel.

III. Bayesian Nonparametric Inverse Reinforcement Learning

Of the many IRL algorithms developed [2, 11, 15, 23–26], most attempt to find a single monolithic reward function that explains the entirety of the observed demonstration set. This reward function must then be necessarily complex in order to explain the data sufficiently, especially when the task being demonstrated is itself complicated. Searching for a complex reward function is fundamentally difficult for two reasons. First, as the complexity of the reward model increases, so too does the number of free parameters needed to describe the model. Thus the search is over a larger space of candidate functions. Second, the process of testing candidate reward functions requires solving for the MDP value function, the computational cost of which typically scales poorly with the size of the MDP state space, even for approximate solutions

[27]. Thus finding a single, complex reward function to explain the observed demonstrations requires searching over a large space of possible solutions and substantial computational effort to test each candidate.

One potential solution to these problems would be to partition the observations into sets of smaller sub-demonstrations. Then, each sub-demonstration could be attributed to a smaller and less-complex class of reward functions. However, such a method would require manual partitioning of the data into an unknown number of groups, and inferring the reward function corresponding to each group. A main contribution of this section is to present an IRL algorithm that automates this partitioning process using Bayesian nonparametric methods. Instead of finding a single, complex reward function the demonstrations are partitioned and each partition is explained with a simple reward function. Note that the assumptions and data format for this approach are substantially different from subgoal discovery methods that learn when given a reward function [17], partition trajectories based on continuous dynamics without regard to changes in the policy [18], learn multiple reward functions for different types of users [47, 48], or IRL methods that find subgoals based on persistent relational factors [19].

In the subgoal learning method below, a generative model is assumed in which these simple reward functions can be interpreted as *subgoals* of the demonstrator. The generative model utilizes a Chinese Restaurant Process (CRP) prior over partitions so that the number of partitions (and thus subgoals) need not be specified *a priori* and can be potentially infinite.

A. Subgoal Reward and Likelihood Functions

The following describes the Bayesian nonparametric subgoal IRL algorithm. We start with two basic definitions.

Definition 1. *A state subgoal g is simply a single coordinate $g \in S$ of the MDP state space. Let c denote a positive constant. The associated state subgoal reward function $R_g(s)$ is $R_g(s) = c\mathbb{I}_g(s)$, where the indicator $\mathbb{I}_g(s)$ is one if $s = g$ and zero otherwise.*

While the notion of a state subgoal and its associated reward function may seem trivial, a more general *feature* subgoal will be defined in the following sections to extend the algorithm to a feature representation of the state space.

Definition 2. *An agent in state s_i moving towards some state subgoal g chooses an action a_i with the following probability:*

$$\pi(a_i|s_i, g) = e^{\alpha Q^*(s_i,a_i,R_g)}(\sum_a e^{\alpha Q^*(s_i,a,R_g)})^{-1} \quad (9)$$

Thus π defines a stochastic policy as in [43], and is essentially our model of rationality for the demonstrating agent. This is the same rationality model as in [26] and [49] and aims to model goal directed behavior without placing all probability on the optimal selection. In Bayesian terms, it defines the likelihood of observed action a_i when the agent is in state s_i. The hyper-parameter α represents our degree of confidence in the demonstrator's ability to maximize reward.

B. Generative Model

The set of observed state-action pairs $\boldsymbol{O}$ defined by (5) are assumed to be generated by the following model. The model is based on the likelihood function (9), but adds a CRP partitioning component. This addition reflects our basic assumption that the demonstrations can be explained by partitioning the data and finding a simple reward function for each partition.

An agent finds itself in state s_i. In analogy to the CRP mixture described in Section II-C, the agent chooses which partition a_i should be added to, where each existing partition j has its own associated subgoal g_j. The agent can also choose to assign a_i to a new partition whose subgoal will be drawn from the base distribution $P(g)$ of possible subgoals. The assignment variable z_i is set to denote that the agent has chosen partition z_i, and thus subgoal g_{z_i}. As in (6), $P(z_i|z_{1:i-1}) = CRP(\eta, z_{1:i-1})$. Now that a partition (and thus subgoal) has been selected for a_i, the agent generates the action according to the stochastic policy $a_i \sim \pi(a_i|s_i, g_{z_i})$ from (9). The joint probability over $\boldsymbol{O}$, $\boldsymbol{z}$, and $\boldsymbol{g}$ is given below, since it will be needed to derive the conditional distributions necessary for sampling:

$$P(\boldsymbol{O}, \boldsymbol{z}, \boldsymbol{g}) = P(\boldsymbol{O}|\boldsymbol{z}, \boldsymbol{g})P(\boldsymbol{z}, \boldsymbol{g}) = P(\boldsymbol{O}|\boldsymbol{z}, \boldsymbol{g})P(\boldsymbol{z})P(\boldsymbol{g}) \quad (10)$$

$$= \prod_{i=1}^{N} \underbrace{P(o_i|g_{z_i})}_{\text{likelihood}} \underbrace{P(z_i|z_{-i})}_{\text{CRP}} \prod_{j=1}^{J_N} \underbrace{P(g_j)}_{\text{prior}} \quad (11)$$

where (10) follows since subgoal parameters g_j for each new partition are drawn independently from prior $P(g)$. As shown in (11), there are three key elements to the joint probability. The likelihood term is the probability of taking each action a_i from state s_i given the associated subgoal g_{z_i}, and is defined in (9). The CRP term is the probability of each partition assignment z_i given by (6). The prior term is the probability of each partition's subgoal (J_N is used to indicate the number of partitions after observing N data-points). The subgoals are drawn i.i.d. from discrete base distribution $P(g)$ each time a new partition is started, and thus have non-zero probability given by $P(g_j)$.

The model assumes that o_i is conditionally independent of o_j for $i \neq j$ given g_{z_i}. Also, it can be verified that the CRP partition probabilities $P(z_i|z_{-i})$ are exchangeable. Thus, the model implies that the data $\boldsymbol{O}$ are exchangeable [45]. Note that this is weaker than implying that the data are independent and identically distributed (i.i.d.). The generative model instead assumes that there is an underlying grouping structure that can be exploited in order to decouple the data and make posterior inference feasible.

The CRP partitioning allows for an unknown and potentially infinite number of subgoals. By construction, the CRP has "built-in" complexity control, i.e., its concentration hyper-parameter η from (6) can be used to make a smaller number of partitions more likely.

C. Inference

The generative model (11) has two sets of hidden parameters, namely the partition assignments z_i for each observation o_i, and the subgoals g_j for each partition j. Thus the job of the

IRL algorithm will be to infer the posterior over these hidden variables, $P(\boldsymbol{z}, \boldsymbol{g}|\boldsymbol{O})$. While both $\boldsymbol{z}$ and $\boldsymbol{g}$ are discrete, the support of $P(\boldsymbol{z}, \boldsymbol{g}|\boldsymbol{O})$ is combinatorially large (since $\boldsymbol{z}$ ranges over the set of all possible partitions of N integers), so exact inference of the posterior is not feasible. Instead, approximate inference techniques must be used. Gibbs sampling [50] is in the family of Markov chain Monte Carlo (MCMC) sampling algorithms and is commonly used for approximate inference of Bayesian nonparametric mixture models [51, 52].

Each Gibbs iteration involves sampling from the conditional distributions of each hidden variable given all of the other variables (i.e., sample one unknown at a time with all of the others fixed). Thus the conditionals for each partition assignment z_i and subgoal g_j must be derived. The conditional for partition assignment z_i is as follows:

$$P(z_i|z_{-i}, \boldsymbol{g}, \boldsymbol{O}) \propto P(z_i, o_i \mid z_{-i}, \boldsymbol{O}_{-i}) \tag{12}$$
$$= P(z_i|z_{-i}, \boldsymbol{g}, \boldsymbol{O}_{-i}) P(o_i|z_i, z_{-i}, \boldsymbol{g}, \boldsymbol{O}_{-i}) \tag{13}$$
$$= P(z_i|z_{-i}) \;\; P(o_i|z_i, z_{-i}, \boldsymbol{g}, \boldsymbol{O}_{-i}) \tag{14}$$
$$= \underbrace{P(z_i|z_{-i})}_{\text{CRP}} \;\; \underbrace{P(o_i|g_{z_i})}_{\text{likelihood}} \tag{15}$$

where (12) is the definition of conditional probability, (13) applies the chain rule, (14) follows from the fact that assignment z_i depends only on the other assignments z_{-i}, and (15) follows from the fact that each o_i depends only on its assigned subgoal g_{z_i}. When sampling from (15), the exchangeability of the data is utilized to treat z_i as if it was the last point to be added. Probabilities (15) are calculated with z_i being assigned to each existing partition, and for the case when z_i starts a new partition with subgoal drawn from the prior $P(g)$. While the number of partitions is potentially infinite, there will always be a finite number of groups when the length of the data N is finite, so this sampling step is always feasible.

Similarly, the conditional for each partition's subgoal g_j is:

$$P(g_j|\boldsymbol{z}, \boldsymbol{O}) \propto P(\boldsymbol{O}_{I_j}|g_j, \boldsymbol{z}, \boldsymbol{O}_{-I_j}) P(g_j|\boldsymbol{z}, \boldsymbol{O}_{-I_j}) \tag{16}$$
$$= \sum_{i \in I_j} P(o_i|g_{z_i}) \; P(g_j|\boldsymbol{z}, \boldsymbol{O}_{-I_j}) \tag{17}$$
$$= \sum_{i \in I_j} \underbrace{P(o_i|g_{z_i})}_{\text{likelihood}} \; \underbrace{P(g_j)}_{\text{prior}} \tag{18}$$

where (16) applies Bayes' rule, (17) follows from the fact that each o_i depends only on its assigned subgoal g_{z_i}, and (18) follows from the fact that the subgoal g_j of each partition is drawn i.i.d. from the prior over subgoals. The index set $I_j = \{i : z_i = j\}$.

Sampling from (18) depends on the form of the prior over subgoals $P(g)$. When the subgoals are assumed to take the form of *state subgoals* (Definition 1), then $P(g)$ is a discrete distribution whose support is the set S of all states of the MDP. The following simplifying assumption is proposed to increase the efficiency of the sampling process.

Proposition 1. *The prior $P(g)$ is assumed to have support only on the set S_O of MDP states, where $S_O = \{s \in S : s = s_i$ for some observation $o_i = (s_i, a_i)\}$.*

This proposition assumes that the set of all possible subgoals is limited to only those states encountered by the demonstrator. Intuitively it implies that during the demonstration, the demonstrator achieves each of his subgoals. This is not the same as assuming a perfect demonstrator (the expert is not assumed to get to each subgoal optimally, just eventually). Sampling of (18) now scales with the number of unique states in the observation set $\boldsymbol{O}$. While this proposition may seem limiting, the simulation results in Section III-F indicate that it does not affect performance compared to other IRL algorithms and greatly reduces the required amount of computation. An alternative to making the assumption in Proposition 1 would be to provide a set of candidate reward functions as input to the algorithm.

Algorithm 1 Bayesian nonparametric IRL

1: **function** BNIRL(MDP/R, Obs. $\boldsymbol{O}$, Conf. α, Conc. η)
2: **for** each unique $s_i \in \boldsymbol{O}$ **do**
3: Compute $V^*(R_g)$, where $g = s_i$ and $R_g = c\mathbb{I}_g(s)$
4: Sample initial subgoal $g_1^{(0)}$ from prior $P(g)$
5: Assign all $z_i^{(0)} = 1$
6: **end for**
7: **while** iteration $t < t_{\text{max}}$ **do**
8: **for** each current subgoal $g_j^{(t-1)}$ **do**
9: Sample subgoal $g_j^{(t)}$ from (18)
10: **end for**
11: **for** each observation $o_i \in \boldsymbol{O}$ **do**
12: **for** each current subgoal $j^{(t)}$ **do**
13: $p(z_i = j|\boldsymbol{z}, \boldsymbol{O}, R_j) \leftarrow$ Prob. of subgoal j, (15)
14: **end for**
15: $p(z_i = k|\boldsymbol{z}, \boldsymbol{O}, R_k) \leftarrow$ Prob. of new subgoal R_k drawn from $P(g)$
16: $z_i^{(t)} \leftarrow$ Sample assignment from normalized probabilities in lines 13–15
17: **end for**
18: **end while**
19: **return** samples $z^{(1:t_{\text{max}})}, g^{(1:t_{\text{max}})}$, discard burn-in/lag ones
20: **end function**

Algorithm 1 defines the proposed Bayesian nonparametric inverse reinforcement learning method. The algorithm outputs samples which form a Markov chain whose stationary distribution is the posterior, so that sampled assignments $\boldsymbol{z}^{(T)}$ and subgoals $\boldsymbol{g}^{(T)}$ converge to a sample from the true posterior $P(\boldsymbol{z}, \boldsymbol{g}|\boldsymbol{O})$ as $T \to \infty$ [50, 53]. Note that instead of solving for the MDP value function in each iteration (as is typical with IRL algorithms), Algorithm 1 pre-computes all of the necessary value functions. The number of required value functions is upper bounded by the number of elements in the support of the prior $P(g)$. When we assume Proposition 1, then the support of $P(g)$ is limited to the set of unique states in the observations $\boldsymbol{O}$. Thus the required number of MDP solutions scales with the size of the observed data set $\boldsymbol{O}$, *not* with the number of required iterations.

D. Convergence in Expected $0-1$ *Loss*

To demonstrate convergence, it is common in IRL to define a loss function which in some way measures the difference

between the demonstrator and the predictive output of the algorithm [23, 25, 26]. In Bayesian nonparametric IRL, the assignments $\boldsymbol{z}$ and subgoals $\boldsymbol{g}$ represent the hidden variables of the demonstration that must be learned. Since these variables are discrete, a $0-1$ loss function is suitable:

$$\mathcal{L}\left[(\boldsymbol{z},\boldsymbol{g}),(\widehat{\boldsymbol{z}},\widehat{\boldsymbol{g}})\right] = \begin{cases} 1 & \text{if } (\widehat{\boldsymbol{z}},\widehat{\boldsymbol{g}}) = (\boldsymbol{z},\boldsymbol{g}) \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

The loss function evaluates to 1 if the estimated parameters $(\widehat{\boldsymbol{z}},\widehat{\boldsymbol{g}})$ are exactly equal to the true parameters $(\boldsymbol{z},\boldsymbol{g})$, and 0 otherwise. We would like to show that, for the Bayesian nonparametric IRL algorithm (Algorithm 1), the expected value of the loss function (19) given a set of observations $\boldsymbol{O}$ is minimized as the number of iterations T increases. Theorem 1 establishes this.

Theorem 1. *Assuming observations $\boldsymbol{O}$ are generated according to the generative model defined by (11), the expected 0–1 loss defined by (19) is minimized by the empirical mode of the samples $(\boldsymbol{z}^{(1:T)},\boldsymbol{g}^{(1:T)})$ output by Algorithm 1 as the number of iterations $T \to \infty$.*

Proof. It can be verified that the maximum *a posteriori* (MAP) parameter values, defined here by $(\widehat{\boldsymbol{z}},\widehat{\boldsymbol{g}}) = \operatorname{argmax}_{(\boldsymbol{z},\boldsymbol{g})} P(\boldsymbol{z},\boldsymbol{g}|\boldsymbol{O})$, minimize the expected 0–1 loss defined in (19) given the observations $\boldsymbol{O}$ (see [54]). By construction, Algorithm 1 defines a Gibbs sampler whose samples $(\boldsymbol{z}^{(1:T)},\boldsymbol{g}^{(1:T)})$ converge to samples from the true posterior $P(\boldsymbol{z},\boldsymbol{g}|\boldsymbol{O})$ so long as the Markov chain producing the samples is ergodic [50]. A sufficient condition for ergodicity of the Markov chain in Gibbs sampling requires only that the conditional probabilities used to generate samples are non-zero [55]. For Algorithm 1, these conditionals are defined by (15) and (18). Since clearly the likelihood (9) and CRP prior (6) are always non-zero, then the conditional (15) is always non-zero. Furthermore, the prior over subgoals $P(g)$ is non-zero for all possible g by assumption, so that (18) is non-zero as well.

Thus the Markov chain is ergodic and the samples $(\boldsymbol{z}^{(1:T)},\boldsymbol{g}^{(1:T)})$ converge to samples from the true posterior $P(\boldsymbol{z},\boldsymbol{g}|\boldsymbol{O})$ as $T \to \infty$. By the strong law of large numbers, the empirical mode of the samples, defined by $(\widetilde{\boldsymbol{z}},\widetilde{\boldsymbol{g}}) = \operatorname{argmax}_{(\boldsymbol{z}^{(1:T)},\boldsymbol{g}^{(1:T)})} P(\boldsymbol{z},\boldsymbol{g}|\boldsymbol{O})$ converges to the true mode $(\widehat{\boldsymbol{z}},\widehat{\boldsymbol{g}})$ as $T \to \infty$, and this is exactly the MAP estimate of the parameters which was shown to minimize the 0–1 loss. □

The rate at which the loss function decreases relies on the rate the empirical sample mode(s) converges to the true mode(s) of the posterior. This is a property of the approximate inference algorithm and, as such, is beyond the scope of this paper (convergence properties of the Gibbs sampler have been studied, for instance in [56]). As will be seen empirically in Section III-F, the number of iterations required for convergence is typically similar to (or less than) that required for other IRL methods.

E. Extension to General Linear Reward Functions

Linear combinations of state features are commonly used in reinforcement learning to approximately represent the value function in a lower-dimensional space [27, 43]. Formally, a k-dimensional feature vector is a mapping $\Phi : S \mapsto \mathbb{R}^k$. Likewise, a discrete k-dimensional feature vector is a mapping $\Phi : S \mapsto \mathbb{Z}^k$, where $\mathbb{Z}$ is the set of integers. Many of the IRL algorithms listed in Section II-B assume that the reward function can be represented as a linear combination of features. We extend Algorithm 1 to accommodate discrete feature vectors by defining a *feature subgoal* in analogy to the state subgoal from Definition 1.

Definition 3. *Given a k-dimensional discrete feature vector Φ, a feature subgoal $g(\boldsymbol{f})$ is the set of states in S which map to the coordinate $\boldsymbol{f}$ in the feature space. Formally, $g(\boldsymbol{f}) = \{s \in S : \Phi(s) = \boldsymbol{f}\}$ where $\boldsymbol{f} \in \mathbb{Z}^k$. The associated feature subgoal reward function is defined as $R_{g(\boldsymbol{f})}(s) = c\mathbb{I}_{g(f)}(s)$, where c is a positive constant and the indicator $\mathbb{I}_{g(f)}(s)$ is one if $s \in g(\boldsymbol{f})$ and is zero otherwise.*

From this definition it can be seen that a state subgoal is simply a specific instance of a feature subgoal, where the features are binary indicators for each state in S. Algorithm 1 runs exactly as before, with the only difference being that the support of the prior over reward functions $P(g)$ is now defined as the set of unique feature coordinates induced by mapping S through Φ. Proposition 1 is also still valid should we wish to again limit the set of possible subgoals to only those feature coordinates in the observed demonstrations, $\Phi(s_{1:N})$. Finally, feature subgoals do not modify any of the assumptions of Theorem 1, thus convergence is still attained in $0-1$ loss.

F. Simulation Results

Simulation results are given for three test cases. All three use a 20×20 Grid World MDP (total of 400 states) with walls. Note that while this is a relatively simple MDP, it is similar in size and nature to experiments done in the seminal papers of each of the compared algorithms. Also, the intent of the experiments is to compare basic properties of the algorithms in nominal situations (as opposed to finding the limits of each).

The agent can move in all eight directions or choose to stay. Transitions are noisy, with probability 0.7 of moving in the chosen direction and probability 0.3 of moving in an adjacent direction. The discount factor $\gamma = 0.99$, and value iteration is used to find the optimal value function for all of the IRL algorithms tested. The demonstrator in each case makes optimal decisions based on the true reward function. While this is not required for Bayesian nonparametric IRL, it is an assumption of one of the other algorithms tested [11]. In all cases, the $0-1$ policy loss function is used to measure performance. The $0-1$ policy loss simply counts the number of times that the learned policy (i.e., the optimal actions given the learned reward function) does not match the demonstrator over the set of observed state-action pairs.

1) Grid World Example: The first example uses the state-subgoal Bayesian nonparametric IRL algorithm. The prior over subgoal locations is chosen to be uniform over states visited by the demonstrator (as in Proposition 1). The demonstrator chooses optimal actions towards each of three subgoals $(x,y) = \{(10,12),(2,17),(2,2)\}$, where the next subgoal is chosen only after arrival at the current one. Figure 1 shows the state-action pairs of the demonstrator (left), the

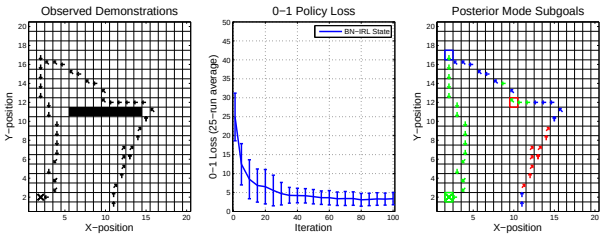


Fig. 1: Observed state-action pairs for simple grid world example (left). Arrows indicate direction of the chosen action and X's indicate choosing the "stay" action. $0-1$ policy loss for Bayesian nonparametric IRL (middle). Posterior mode of subgoals and partition assignments (right). Colored arrows denote assignments to the corresponding colored boxed subgoals.

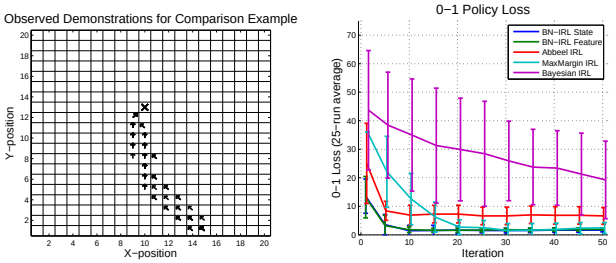


Fig. 2: Observed state-action pairs for grid world comparison example (left). $0-1$ policy loss comparison for various IRL algorithms (right).

$0-1$ policy loss averaged over 25 runs (center), and the posterior mode of subgoals and partition assignments (colored arrows denote assignments to the corresponding colored boxed subgoals) after 100 iterations (right). The algorithm reaches a minimum in loss after roughly 40 iterations, and the mode of the posterior subgoal locations converges to the correct coordinates. We note that while the subgoal locations have correctly converged after 100 iterations, the partition assignments for each state-action pair have not yet converged for actions whose subgoal is somewhat ambiguous, mainly because CRP makes no assumptions on the temporal relationship between subgoals.

2) Grid World with Features Comparison: In the next test case, Bayesian nonparametric IRL (for both state- and feature-subgoals) is compared to three other IRL algorithms, using the same Grid World setup as in Section III-F1: "Abbeel" IRL using the quadratic program variant [11], Maximum Margin Planning using a loss function that is non-zero at states not visited by the demonstrator [23], and Bayesian IRL [26]. A set of six features $\Phi_{1:6}(s)$ are used, where feature k has an associated state s_{Φ_k}. The value of feature k at state s is simply the Manhattan distance (1-norm) from s to s_{Φ_k}; i.e., $\Phi_k(s) = ||s - s_{\Phi_k}||_1$. The true reward function is defined as $R(s) = \boldsymbol{w}^T \boldsymbol{\Phi}(s)$ where $\boldsymbol{w}$ is a vector of randomly-chosen weights. The observations consist of five demonstrations starting at state $(x, y) = (15, 1)$, each having 15 actions which follow the optimal policy corresponding to the true reward function. Note that this dataset satisfies the assumptions of the three compared algorithms, though it does not strictly follow the generative process of Bayesian nonparametric IRL. Figure 2 shows the state-action pairs of the demonstrator (left) and the $0-1$ policy loss, averaged over 25 runs versus iteration for each algorithm (right). All but Bayesian IRL achieve convergence to the same minimum in policy loss by 20 iterations, and Bayesian IRL converges at roughly 100 iterations (not shown). Even though the assumptions of Bayesian nonparametric IRL were not strictly satisfied (the assumed model (11) did not generate the data), both the state- and feature-subgoal variants of the algorithm achieved performance comparable to the other IRL methods.

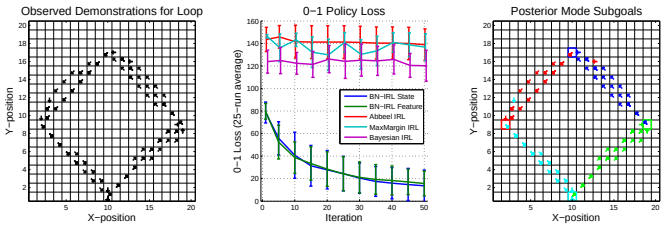


Fig. 3: Observed state-action pairs for grid world loop example (top left). Comparison of $0-1$ Policy loss for various IRL algorithms (top right). Posterior mode of subgoals and partition assignments (bottom).

3) Grid World with Loop Comparison: In the final experiment, five demonstrations are generated using subgoals as in Section III-F1. The demonstrator starts in state $(x, y) = (10, 1)$, and proceeds to subgoals $(x, y) = \{(19, 9), (10, 17), (2, 9), (10, 1)\}$. Distance features (as in Section III-F2) are placed at each of the four subgoal locations. Figure 3 (left) shows the observed state-action pairs. This dataset clearly violates the assumptions of all three of the compared algorithms, since more than one reward function is used to generate the state-action pairs. However, the assumptions are violated in a reasonable way. The data resemble a common robotics scenario in which an agent leaves an initial state, perform some tasks, and then returns to the same initial state.

Figure 3 (center) shows that the three compared algorithms, as expected, do not converge in policy loss. Both Bayesian nonparametric algorithms, however, perform almost exactly as before and the mode posterior subgoal locations converge to the four true subgoals (Figure 3 right). Again, the three compared algorithms would have worked properly if the data had been generated by a single reward function, but such a reward function would have to be significantly more complex (i.e., by including temporal elements). Bayesian nonparametric IRL is able to explain the demonstrations without modification or added complexity.

G. Comparison of Computational Complexities

BNIRL has two stages of computation:

- In the initialization stage, optimal action-value functions are computed for each candidate reward state, i.e., for each unique demonstration state by Proposition 1. Since many methods exist for finding optimal action-value functions [27, 43], the computational complexity of the operation is denoted as $O(\text{MDP})$.
- In the sampling stage, each iteration requires assigning observations to a subgoal reward. The complexity of each sampling iteration is $O(N_{\text{obs}}\ \eta \log N_{\text{obs}})$, where N_{obs} is the number of observed state-action pairs in the demonstration,

η is the CRP concentration parameter, and $\eta \log N_{\text{obs}}$ is the expected number of active subgoals in the CRP.

The overall complexities of each stage are not directly comparable since results on the number of iterations required for Gibbs sampling convergence are not well established [56]. However, in practice the first stage (calculating optimal action-value functions) dominates the complexity of the overall algorithm.

As outlined below, other IRL algorithms calculate optimal action value functions once per iteration. Since BNIRL calculates the optimal action value function at a maximum of once per demonstration state, a rough complexity comparison can be made by comparing the number of times the MDP must be solved for each algorithm. The following summarizes a complexity analysis given in each respective original work:

1) **Abbeel IRL [57]:** An upper bound on the number of iterations required to guarantee feature count matching error $\frac{\epsilon}{4}$ is given as $\frac{48k}{(1-\gamma)^2\epsilon^2}$, where k is the dimension of the state space and γ is the MDP discount factor. Each iteration requires computation of an optimal action-value function.

2) **Bayesian IRL [26]:** The number of iterations required is related to the mixing time of the MCMC method used. The chain is said to "rapidly mix" in $O(n^2d^2\alpha^2e^{2\beta}\log\frac{1}{\epsilon})$ sampling iterations, where n is the dimension of the state space, d is a bound on the reward magnitude, and α, β and ϵ are parameters. Each sampling iteration requires computation of an optimal action-value function.

3) **MaxMargin IRL [23]:** While no analytical expression is given, convergence is said to be sub-linear for a diminishing step-size rule which achieves a minimum in error under a strong convexity assumption. As in the above two algorithms, each iteration requires computation of an optimal value function.

Thus, the effective complexity of these algorithms (number of optimal value functions that must be computed) scales with the number of iterations needed for convergence. As shown above, the number of required iterations can depend on many parameters of each algorithm. In BNIRL, the effective complexity is upper-bounded by the number of unique states in the demonstration, which highlights a fundamental computational difference of BNIRL versus previous methods.

To give an empirical sense of computation times for the example in Section III-F2, Table I compares average initialization and per-iteration run-times for each algorithm. These are given only to show general trends, as the Matlab implementations of the algorithms were not optimized for efficiency. The initialization of BNIRL takes much longer than the others, since during this period the algorithm is pre-computing optimal value functions for each of the possible subgoal locations (i.e., each of the states encountered by the demonstrator). However, the BNIRL per-iteration time is roughly an order of magnitude less than the others, since they must re-compute an optimal value function each iteration.

The example which generated the data in Table I was selected for BNIRL to perform comparably in overall runtime to Abbeel IRL such that a fair comparison of initialization versus per-iteration runtimes can be made. This selection highlights the fundamental performance tradeoff between BNIRL and the other IRL methods compared. By Proposition 1, BNIRL limits the candidate subgoals to the states observed in the demonstration. This proposition limits the potential complexity of the reward representation, but it also places an upper-bound on the number of value functions that must be calculated. In the compared IRL methods, the reward function is parametrized and the algorithms iteratively search over a continuous parameter space, computing a new value function at each iteration. In this case, no assumption is made about the number of candidate reward functions (other than the reward parametrization itself) at the cost of an asymptotic number of value functions to be computed.

TABLE I: Run-time comparison for various IRL algorithms.

	Initialize (sec)	Per-iter. (sec)	Iter. to Converge	Total (sec)
BNIRL	15.3	0.21	10	17.4
Abbeel-IRL	0.42	1.65	10	16.9
MaxMargin-IRL	0.41	1.16	20	23.6
Bayesian-IRL	0.56	3.27	105	344

As a result of this fundamental difference in algorithmic structure, there are scenarios when BNIRL will perform computationally faster than the other methods, and vice versa. In cases where the demonstration set is small and there are a large number of demonstrator subgoals, BNIRL will generally execute faster since its computation scales with the number of unique demonstration states and it has the ability to learn multiple subgoal reward functions. The other IRL methods will generally execute slower in this case, since they must search for a more complex representation. In cases where there is a large amount of demonstration data and there are not multiple subgoals, BNIRL will generally execute slower since it must find a value function for each unique demonstration state. The other IRL will generally execute faster in this case, since their computation does not scale with demonstration size and a less-complex reward representation is required.

H. Limitations of the Learned Subgoal Reward Function

The use of state and feature subgoal reward functions is limiting in that the resulting reward learning process is similar to trajectory segmentation. This is a direct result of the proposed set of candidate rewards, which is given to be set of states (or features) encountered in the trajectory. Even so, as opposed to trajectory segmentation or policy learning methods, BNIRL is a true reward learning method whereby reward functions are posed and tested. While relatively simple subgoal reward functions are presented here, more complex classes of reward function can be learned using BNIRL. The main requirement for a new class of reward functions is that a set of candidate rewards be tractably posed. In this case, the set of candidate rewards comes from states (or features) encountered in the demonstration. However, any method which is able to take demonstration trajectories and generate a set of candidate reward functions is applicable. Extending the algorithm to learn more general classes of reward functions is an area of potential future work.

Another limitation of the subgoals (or, more generally, reward functions) generated by BNIRL is that the temporal

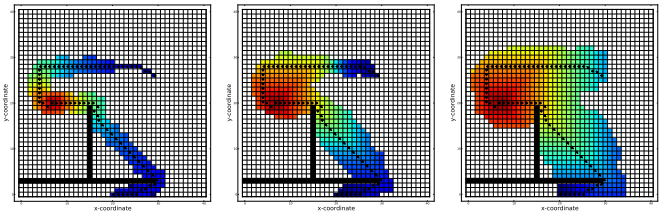


Fig. 4: Progression of real-time dynamic programming [40] sample states for the Grid World example. The algorithm starts with the initial set (top) based on the demonstration set (denoted by arrows), and uses greedy simulations to progressively expand the set of sample states (middle and bottom) over which value iteration is performed.

sequence is not explicitly learned. Thus, using the learned reward functions to re-create demonstration sequences requires additional post-processing by a planning algorithm. While BNIRL is purely a reward learning algorithm, planning and control algorithms can be applied which utilize the learned reward functions to create macro-actions to achieve an overall goal. As an example, Section V-F demonstrates how learned subgoals can be utilized in the MDP option framework for overall task planning. The experimental results in Section VI-B demonstrate the execution of the learned options, though more complex sequencing and planning using options is beyond the scope of the paper and left as future work.

IV. Approximations to the Demonstrator Likelihood

This section presents simulation results which apply *(i)* Real-time Dynamic Programming (RTDP) and *(ii)* "action comparison" as two action likelihood approximation methods to handle challenging problems with large and continuous domains. A detailed explanation and implementation of these methods can be found in [58] and [42].

As an example of using RTDP in BNIRL, consider the Grid World domain shown in Figure 4. The agent can move in all eight directions or choose to stay. Transitions are noisy with probability 0.7 of moving in the chosen direction, and the discount factor $\gamma = 0.99$. The demonstration set $\boldsymbol{O}$ is denoted by arrows, indicating actions chosen from each state. The initial set of RTDP sample states $\tilde{S}$ is chosen to be the set of states encountered in the demonstration $\boldsymbol{O}$, as well as any other states reachable in one transition. Value iteration is performed on these states for an example candidate reward function, and the resulting value function is shown in Figure 4a. A random state $s \in \boldsymbol{O}_i$ is then chosen, and a greedy simulation is performed. All states encountered during the simulation (as well as any other states reachable in one transition) are added to sample states denoted by $\tilde{S}$ [58] . The cycle repeats, and Figures 4b and 4c shows the progression of sample states and corresponding value functions. The process terminates when the greedy simulation fails to add any new states to $\tilde{S}$.

In order to test the scalability of the RTDP Q^* approximation, the CPU run-times of five different methods are compared: BNIRL using full value iteration, Abbeel IRL (from [11], a representative conventional IRL method) using full value iteration, BNIRL using RTDP, Abbeel IRL using RTDP,

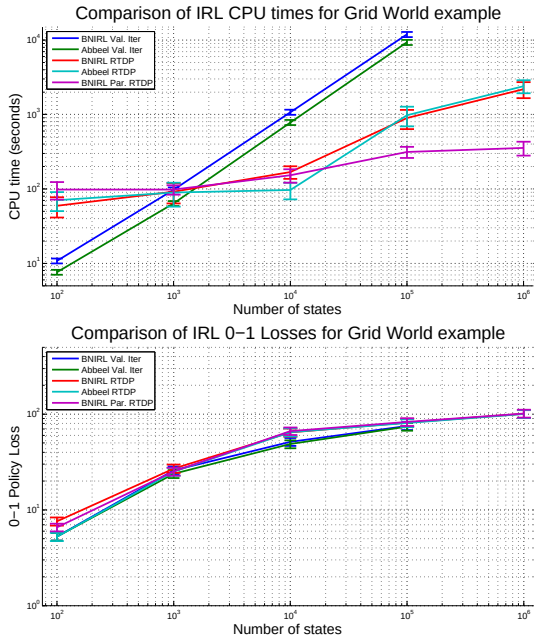


Fig. 5: Comparison of average CPU runtimes for various IRL algorithms for the Grid World example (left), along with the average corresponding 0-1 policy loss (right). Both plots use a log-log scale, and averages were take over 30 samples.

and BNIRL using parallelized RTDP. In the parallel BNIRL case, the pre-computation of the required approximate value functions is done on a cluster of 25 computing cores. The ability to compute value functions in parallel is a feature of the BNIRL algorithm since the set of reward function candidates is guaranteed to be finite by Proposition 1. Abbeel IRL (as well as other conventional IRL methods [11, 15, 23–26]) cannot be parallelized due to the fact that a new value function must be computed at each sequential iteration. Computation is performed on a Pentium i7 3.4GHz processor with 8GB RAM. Implementations of each algorithm have not been optimized for speed, but computation time results are still meaningful for evaluating order-of-magnitude trends.

Figure 5a shows average CPU run-times of each method (lower is better) for Grid World domains ranging from 10^2 to 10^6 states. For each domain size, demonstrations are generated with a greedy controller starting and ending at randomly-chosen states. As can be seen, both BNIRL and Abbeel IRL using full value iteration become extremely slow for problems larger than 10^3 states (data points for 10^6 states are not included, as they would take weeks to compute). Methods using RTDP are slower for small problem sizes (this is due to the extra time needed for simulations to expand the set of sample states). However, beyond problems with 10^3 states, RTDP methods are roughly an order of magnitude faster than full value iteration. Finally, the parallelized BNIRL method using RTDP shows significantly faster performance than the non-parallelized version and Abbeel IRL with RTDP, because 25 computing cores can be used in parallel to calculate the necessary value functions, which is unique to BNIRL.

To ensure that the RTDP Q^* approximation does not affect the quality of the learned reward function, Figure 5b shows the average 0-1 policy loss of each algorithm (lower is better) for

each grid size. The 0-1 policy loss simply counts the number of times that the learned policy (i.e., the optimal actions given the learned reward function) does not match the demonstrator over the set of observed state-action pairs. As can be seen, using RTDP to approximate Q^* does not have an adverse effect on reward learning performance, as the loss for the RTDP methods is only slightly higher (no more than 4% in all cases) than the full value iteration methods.

V. Gaussian Process Subgoal Reward Learning

The Bayesian nonparametric inverse reinforcement learning (BNIRL) method developed in Section III is generalized in this section to handle fully continuous demonstration domains by using Gaussian process reward representations and Gaussian process dynamic programming [59] as a method of finding approximate action-value functions. Further, the option MDP (skills) framework [16] enables execution of the learned behaviors by the robotic system and provides a principled basis for future learning and skill refinement. In Section VI, experimental results are given for a robotic car domain, identifying maneuvering skills such as drifting turns, executing the learned skills autonomously, and providing a method for quantifying the relative skill level of the original demonstrator.

A. Gaussian Process Subgoal Reward Learning Algorithm

BNIRL algorithm (Algorithm 1) learns multiple reward functions, but it is only applicable to discrete domains. The method was extended to a continuous domain in [58], [42], but that extension relies on access to an existing closed-loop controller, which may not be available in general. This section presents the GP subgoal reward learning (GPSRL) algorithm, which learns GP subgoal reward representations from unsegmented, continuous demonstration. The GPSRL algorithm assumes two key inputs:

1) Demonstration set of state-action pairs $\boldsymbol{O}=\{(s_i,a_i)\}_{i=1}^{N}$. The continuous demonstration must be measured and down-sampled to the desired time interval. This is not the same as discretization of the continuous state space, it is instead sampling a continuous trajectory at finite time intervals.

2) A state transition function, $s'=f(s,a)$, which models the dynamics that generated the demonstration. This is not a model of the demonstrator, just a model of the state transition *given* an action. If no model is available, many system identification techniques exist which can learn a transition model [60]. The full GPSRL method is shown in Algorithm 2, comprising three main stages that are explained in the subsections below. First, the set of candidate subgoal rewards is constructed (lines 2-7). Next, Gaussian process dynamic programming is used to approximate the optimal action-value function Q^* for each candidate subgoal in parallel (lines 8-10). Finally, approximate posterior inference is done using Gibbs sampling (lines 11-19).

B. Subgoal Reward Representation

Since the demonstration space is assumed to be continuous, a subgoal reward at a single coordinate of the state space (as in BNIRL [41]) is ill-defined. A subgoal reward representation with broader support is achieved using Gaussian processes (GPs). Each subgoal reward is simply a GP with one training point, $GP(s_g,r,k_g)$, where s_g is the subgoal state, r is a positive scalar reward (the magnitude of which is not critical to the algorithm), and $k_g(\cdot,\cdot)$ is a kernel function. The GP spreads the reward to the neighborhood around s_g, according to the kernel function k_g.

Algorithm 2 Gaussian Process Subgoal Reward Learning

1: $\boldsymbol{R}_{sg}, \boldsymbol{S}_{\text{supp}} \leftarrow \{\}$
2: **for** each demonstration state $s_i \in \boldsymbol{O}$ **do**
3: **if** $||s_i - s|| > \epsilon \quad \forall s \in \boldsymbol{S}_{\text{supp}}$ **then**
4: $\boldsymbol{S}_{\text{supp}} \leftarrow \{\boldsymbol{S}_{\text{supp}}, s_i\}$ ▷ Build set of support states for GPDP
5: $\boldsymbol{R}_{sg} \leftarrow \{\boldsymbol{R}_{sg}, GP(s_i, r, k_g)\}$ ▷ Build set of candidate subgoals
6: **end if**
7: **end for**
8: **for** each candidate subgoal $R_j \in \boldsymbol{R}_{sg}$ (in parallel) **do**
9: $\widehat{Q}^*(R_j) \leftarrow \text{GPDP}(S_{\text{supp}}, R_j, f(s'|s,a))$ ▷ Gaussian process DP [59]
10: **end for**
11: **while** iteration $t < N_{\text{iter}}$ **do** ▷ Gibbs sampling of subgoal posterior assignments
12: **for** each observation $o_i \in \boldsymbol{O}$ **do**
13: **for** each current partition $j^{(t)}$ **do**
14: $p(z_i = j|\boldsymbol{z}, \boldsymbol{O}, R_j) \leftarrow$ Prob. of partition j Eq.(22)
15: **end for**
16: $p(z_i = k|\boldsymbol{z}, \boldsymbol{O}, R_k) \leftarrow$ Probability of new partition with R_k drawn from $\boldsymbol{R}_{sg}$
17: $z_i^{(t)} \leftarrow$ Sample assignment from normalized probabilities in lines 13–16
18: **end for**
19: **end while**
20: **return** mode of samples $\boldsymbol{z}^{(1:N_{\text{iter}})}$ & subgoal rewards R_j

As in BNIRL, a key assumption of GPSRL is that the set of possible subgoals comes from the demonstration itself, avoiding *a priori* discretization of the state space to generate a candidate subgoal reward set. The set of possible rewards $\boldsymbol{R}_{sg}$ is thus the set of subgoal rewards corresponding to the sampled demonstration:

$$\boldsymbol{R}_{sg} = \{\ GP(s,r,k_g), \quad \forall s \in \boldsymbol{O}\ \} \tag{20}$$

To avoid redundant subgoals, the set is built incrementally such that a new subgoal is not added if it is ϵ-close to a subgoal already in $\boldsymbol{R}_{sg}$ (lines 2-7 of Algorithm 2). The parameter ϵ is thus chosen to scale the size of the candidate set of subgoal rewards, and correspondingly the computational requirements of the GPSRL algorithm.

C. Action Likelihood

A softmax likelihood based on the optimal Q^* function is used similar to (9):

$$p(O_i|R_j) = p(a_i|s_i,R_j) \propto \exp(\ \alpha Q^*(s_i,a_i|R_j)\ ) \tag{21}$$

The parameter α again represents our confidence in the demonstrator's ability to maximize reward, and further considerations for its selection are given in Section VI-B1.

D. Gaussian Process Dynamic Programming

In order to compute the likelihood value in (21), the optimal Q^* function is necessary. In general, calculating the optimal value function for discrete systems is computationally difficult, and even more so for continuous systems. This necessitates the use of an approximate method, and Gaussian process dynamic programming (GPDP) [59] is chosen for this purpose. GPDP generalizes dynamic programming to continuous domains by representing the value and action-value functions with Gaussian processes. Thus, the algorithm requires only a set of *support states* to train the value function GPs, instead of requiring discretization or feature mapping of the state space.

GPDP is particularly well-suited for the task of approximating the Q^* function in (21) for several reasons. Foremost, the support points used to learn the value function in GPDP come directly from the demonstration, i.e., $\boldsymbol{S}_{\text{support}} = \{s : s \in \boldsymbol{O}\}$. This effectively focuses computational effort only on areas of the state space that are relevant to the reward learning task. Also, the Gaussian process subgoal reward representation from Section V-B is naturally compatible with the GPDP framework. Finally, the output of the GPDP algorithm enables evaluation of the approximated optimal action-value function $\widehat{Q}^*$ at any arbitrary state s via evaluation of a GP mean, allowing for efficient calculation of the likelihood (21). Note that the max in the denominator of (21) must be found numerically by evaluating $\widehat{Q}^*$ at several test actions, which in practice are distributed uniformly between action bounds.

The computational complexity of Algorithm 2 is dominated by the GPDP calculation of $\widehat{Q}^*(R_j)$ for each candidate subgoal reward $R_j \in \boldsymbol{R}_{sg}$ (lines 8-10). However, this is easily parallelized on a computing cluster allowing for substantial savings in computation time.

E. Bayesian Nonparametric Mixture Model and Subgoal Posterior Inference

The GPSRL algorithm learns multiple subgoals reward functions from the demonstration set. To avoid the need to pre-specify or constrain the number of learned subgoals, a Bayesian nonparametric model is used. In the model, each state-action pair in the demonstration $\boldsymbol{O}$ is assigned to a partition. The vector $\boldsymbol{z} \in \mathbb{R}^{|\boldsymbol{O}|}$ stores partition assignments, so that $z_i = j$ implies that observation $O_i = (s_i, a_i)$ is assigned to partition j. Each partition has an associated subgoal from the set of candidate GP subgoal reward functions $\boldsymbol{R}_{sg}$. The posterior probability of assignment z_i to partition j is defined as follows (see [41] for a more detailed derivation):

$$p(z_i = j|\boldsymbol{z}, \boldsymbol{O}, R_j) \;\propto\; \underbrace{p(z_i = j|\boldsymbol{z}_{-i})}_{\text{CRP prior (6)}} \quad \underbrace{p(O_i|R_j)}_{\text{action likelihood (21)}} \tag{22}$$

where R_j is the GP subgoal reward corresponding to partition j. The CRP prior, which encourages clustering into large partitions, is defined by (6). The action likelihood term, which encourages similarity within partitions, is defined by (21).

As in the BNIRL algorithm, Gibbs sampling is used for approximate inference of the Bayesian nonparametric mixture model. The Gibbs sampling procedure comprises lines 11-19 of Algorithm 2.

F. Converting Learned Subgoals to MDP Options

Once subgoal rewards are learned using GPSRL they can be easily cast in the options MDP framework [16]. As summarized in Section II-A, an option is defined by the initiation set I_o, the option policy π_o, and the terminating condition β_o. For a learned subgoal reward R_j centered at subgoal state s_g, the initiation set is defined as those states which are ϵ-close (where ϵ is the parameter from Section V-B) to a demonstration state which is assigned to subgoal R_j by the GPSRL sampling step. Since the approximate optimal action value function $\widehat{Q}^*(s, a|R_j)$ for subgoal reward R_j is already calculated in the GPDP step of GPSRL, the option policy π_o is simply the corresponding optimal policy. Finally, the set of terminating states is simply the set of states which are ϵ-close to s_g:

$$I_o(R_j) \triangleq \{s \in S : ||s - s_i|| < \epsilon, s_i \in \boldsymbol{O}, z_i = j\} \tag{23}$$

$$\pi_o(s|R_j) \triangleq \underset{a}{\text{argmax}} \quad \widehat{Q}^*(s, a|R_j) \tag{24}$$

$$\beta_o(R_j) \triangleq \{s \in S : ||s - s_g|| < \epsilon\} \tag{25}$$

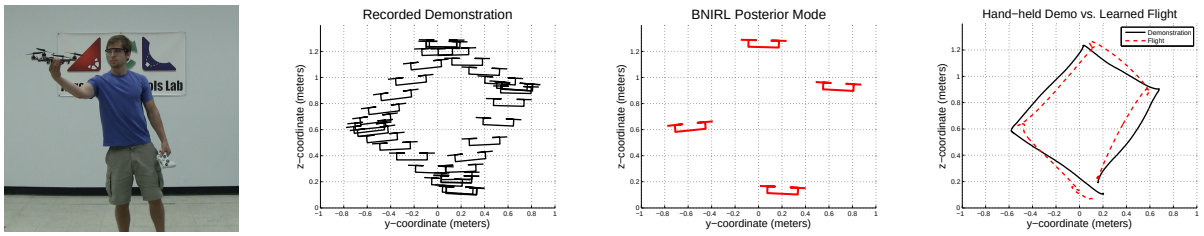
It is worth noting that after identifying subgoals (or more general reward functions, depending on the application), an additional planner is necessary to reproduce the plan over options. The conversion of learned GPSRL subgoals into MDP options is validated experimentally in Section VI, in which RC car driving maneuvers are learned from demonstration and executed by the autonomous system.

VI. Experimental Results

The broad focus of this paper is to enable scalable reward learning from demonstration for real-world robotic systems. This section presents experimental results that validate the use of BNIRL and GPSRL on experimental robotic hardware systems.

A. Learning Quadrotor Flight Maneuvers from Hand-Held Demonstration with Action Comparison

To test the action comparison likelihood approximation described in [42, 58], BNIRL is used to learn quadrotor flight maneuvers from a hand-held demonstration. First, the maneuver is demonstrated by motioning with a disabled quadrotor helicopter (Figure 6a) while the pose and velocities of the quadrotor are tracked and recorded by the motion capture system down-sampled to 20Hz (Figure 6b). In this case, the state vector is 2-dimensional (Y-Z) position and the components of the action vector are the corresponding velocities. For the 2-D quadrotor model in [42, 58], the BNIRL algorithm is used to generate an approximate posterior distribution over the demonstrator's subgoals. Figure 6c shows the mode of the sampled posterior, which converges to four subgoals, one at at each corner of the demonstrated trajectory. The subgoals are then sent as waypoints to an autonomous quadrotor which executes them in actual flight, thus recreating the demonstrated trajectory. Flight tests are conducted in the RAVEN indoor testbed [61] using the flight control law described in [62]. Figure 6d plots the hand-held trajectory against the autonomous flight, showing a qualitative match between the demonstration and the resulting learned behavior.



(a) (b) (c) (d)

Fig. 6: Human demonstrator motions with a disabled quadrotor (a) while a motion capture system records and down-samples demonstration (b). BNIRL algorithm with action comparison likelihood converges to a mode posterior with four subgoals, one at each corner of the demonstrated trajectory (c). Finally, an autonomous quadrotor takes the subgoals as waypoints and flies the learned trajectory (d).

To demonstrate the ability of BNIRL to handle cyclic, repetitive demonstrations, Figure 7 shows a cluttered trajectory where the demonstrator moves randomly between the four corners of a square. Overlaid are the four subgoals of the converged posterior, which correctly identify the four key subgoals inherent in the demonstration.

Figure 8a shows another example in which the demonstrated trajectory is a flip. The BNIRL algorithm using action comparison likelihood converges to posterior subgoals at the bottom and the top of the trajectory, with the quadrotor being inverted at the top (see Figure 8b). The subgoal waypoints are executed by the autonomous flight controller and the actual flight path is overlaid on Figure 8a, again showing the qualitative matching between demonstrated and learned behavior.

The experimental results highlight the ability of the BNIRL algorithm to learn the reward effectively even when the demonstration domain dynamics are different than the dynamics of the autonomous system. The demonstration in this case (hand-held motions) is not necessarily dynamically feasible for the autonomous system, yet the algorithm is still able to learn rewards that generate qualitatively similar behavior. Finally, note that the BNIRL sampling process for the three examples above takes roughly three seconds to converge to the posterior mode on an Intel i7 2.4GHz laptop using an unoptimized implementation. This is due to the fact that evaluation of the closed-loop control action is fast, making BNIRL suitable for online reward learning (see [58], [42] for details).

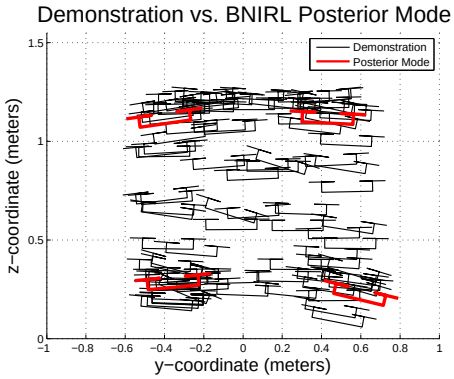
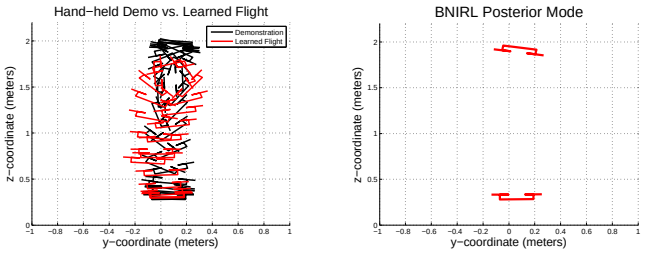


Fig. 7: A cluttered trajectory in which the demonstrator moves randomly between the four corners of a square is shown in black. The BNIRL posterior mode is shown in red, which consists of four subgoals, one at each corner of the square as expected.



(a) (b)

Fig. 8: Hand-held demonstrated quadrotor flip shown in black (a). BNIRL posterior mode converges to two subgoals (bottom and top (inverted) of the flip trajectory) (b). Autonomous quadrotor uses subgoals as waypoints and flies the learned trajectory, (a) in red.

B. Learning Driving Maneuvers from Demonstration with GPSRL

This section presents experimental results demonstrating the ability of GPSRL to learn driving maneuvers for an RC car. Demonstrating and learning such maneuvers is typically challenging due to highly non-linear tire slip dynamics which are difficult to model or predict. The demonstration state vector consists of the body velocities $\dot{x}_b$ and $\dot{y}_b$, heading rate $\dot{\psi}$, and wheel speed ω. Learned subgoals can thus be specified as a GP trained in this 4-dimensional state space. Actions consist of the steer angle δ and commanded wheel speed ω_c (motor torque to the drive wheels is then set by an inner-loop PI controller on wheel speed). Figure 9 shows the RC car used in the experiment along with a diagram of states and actions. Demonstrations are performed via manual remote-controlled operation with a joystick. States $\dot{x}_b, \dot{y}_b$ and $\dot{\psi}$ are measured with a motion capture system sampled at 100Hz, and wheel speed ω is measured onboard with an optical encoder and transmitted wirelessly at 100Hz. The transition model f required for GPDP (Section V-D) is taken from a basic model of car dynamics with wheel slip [63], with model parameters identified from test data.

The squared exponential kernel $k_{\text{SE}}(\boldsymbol{x}, \boldsymbol{x}')$ is used for all GPs, with parameters optimized using gradient ascent to local maxima of the log evidence each time a new GP is created or

modified. The CRP concentration parameter $\eta = 0.001$ is used for all experiments. While detailed discussion of selecting this parameter is omitted, empirical findings indicate that subgoal learning results are not particularly sensitive to η.

Figure 10 (top) shows a 30-second demonstration which includes straight-line driving, standard left/right turns, and advanced drifting turns. The GPSRL algorithm is applied with $\alpha = 25$, and convergence to six learned subgoal rewards is attained within roughly 200 sampling iterations (corresponding to roughly 6 minutes of wall-clock time on an Intel i7 2.4GHz laptop using an unoptimized implementation). Figure 10 (bottom) shows the six subgoal state locations, where arrows represent the body velocities $\dot{x}_b$ and $\dot{y}_b$, the rotation of the rectangle represents heading rate $\dot{\psi}$, and wheel speed ω is omitted for clarity.

The learned subgoals correctly identify the six basic maneuvers from the demonstration: stop, drive straight, left turn, right turn, left drifting turn, and right drifting turn. The trajectory is color-coded to show the partition assignments of the demo states to the six learned subgoals (Figure 10, left). Note that the Bayesian nonparametric model from Section V-E is not biased towards clustering contiguous trajectory segments, yet the posterior mode assignments shows that contiguous segments are indeed clustered into appropriate subgoals, as would be expected.

To explore the behavior of GPSRL as more demonstration data is added, ten more 30-second human-controlled demonstrations were recorded with the demonstrator instructed to include the same types of behavior as in Figure 10. Figure 11 shows the number of subgoals learned as each new demonstration is added, averaged over 25 trials, with the confidence parameter again set to $\alpha = 25$. The number of learned subgoals does not increase arbitrarily with more demonstration data, and stays within two standard deviations of six learned subgoals from the single demonstration case.

1) Confidence Parameter Selection and Expertise Determination: The confidence parameter α has a direct effect on the posterior distribution (22) and thus the number of subgoals learned from a given demonstration. Since α represents the expected degree to which the demonstrator is able to maximize reward, there is thus no analytical method for choosing the parameter. Even so, small or large values of alpha produce consistent trends in the number of learned subgoals. In the limit as $\alpha \to 0$, the demonstrator is assumed to choose arbitrary actions that do not attempt to maximize reward. This leads to the discovery of a single partition since the entire demonstration can be explained as noisy or suboptimal actions towards any arbitrary subgoal. In the limit as $\alpha \to \infty$, the demonstrator is assumed perfectly optimal. This leads to a larger number of learned subgoals since any noise or mistakes present in the demonstrated actions will be treated as completely intentional, and the resultant state will likely be added as a subgoal.

Figure 12 (left) shows a demonstration which consists of a mixture of right-handed turns of three distinct turning radii. This demonstration was intentionally created so that there are three unambiguously "true" subgoals. Figure 12 (right) shows the number of learned subgoals for a logarithmic sweep of

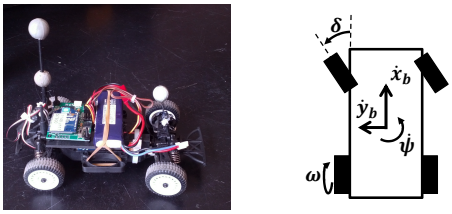


Fig. 9: RC car used for experimental results (left) with optical encoder, battery, radio modem, and reflective markers for motion capture. Diagram of car state (right) with body velocities $\dot{x}_b$ and $\dot{y}_b$, heading rate $\dot{\psi}$, wheel speed ω, and steering command δ.

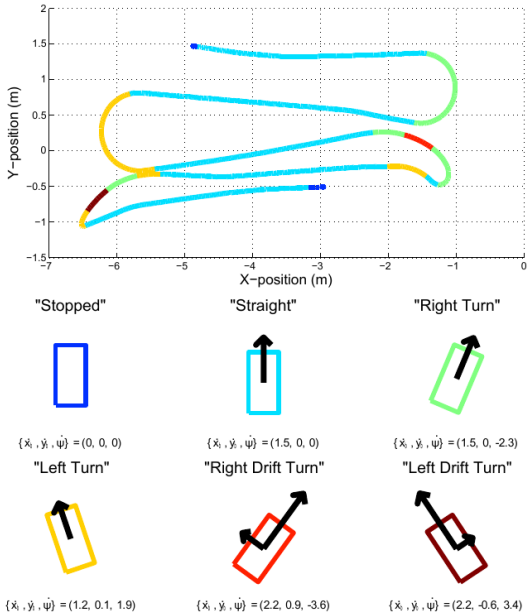


Fig. 10: Thirty-second manually-controlled demonstration trajectory (top) starting in upper-left and ending in lower-middle. Six learned subgoal state locations (bottom), where arrows represent the body velocities $\dot{x}_b$ and $\dot{y}_b$, the rotation of the rectangle represents the heading rate $\dot{\psi}$, and the wheel speed ω is omitted for clarity. Learned subgoal labels ("Left turn", etc.) added manually.

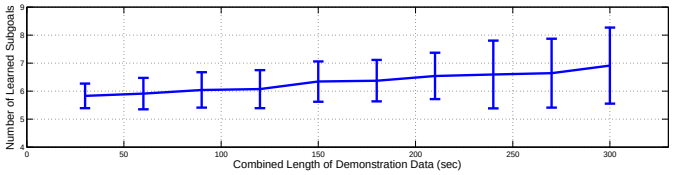


Fig. 11: Number of subgoals learned versus total length of demonstration data sampled, averaged over 25 trials. The number of learned subgoals does not increase arbitrarily with more demonstration data, and stays within 2-σ of the 6 learned subgoals from a single demonstration.

α averaged over 50 trials, with 1500 sampling iterations per trial. As expected, there is a range of $\alpha < 10$ through which only one subgoal is discovered. For $10 < \alpha < 60$ the number of subgoals discovered is within two standard deviations of the true value (three), showing that there is a relatively large range of suitable parameter settings. For $\alpha > 60$ the algorithm

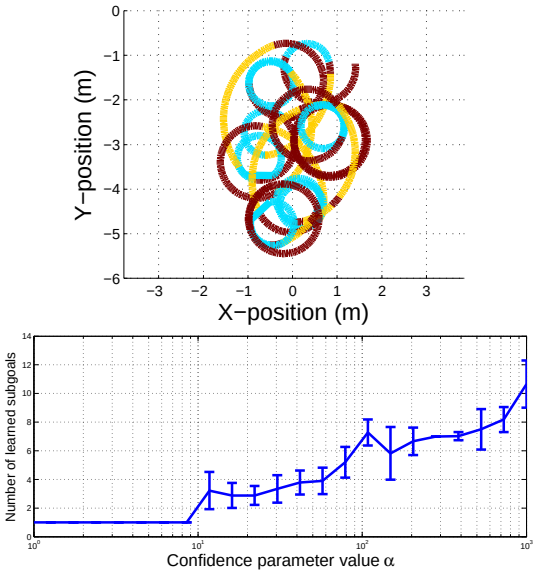


Fig. 12: Demonstration with three distinct right-turning radii (top). Number of learned subgoals (50-trial average) versus the confidence parameter value α (bottom). The trajectory is color-coded with partition labellings for $\alpha = 25$, showing correct subgoal identification for the three turning radii.

discovers more subgoals as expected, since noisy state-actions are interpreted as intentional.

The confidence parameter can also be used in the opposite way to quantify the level of "expertise" of the demonstrator. Consider instead that the demonstration in Figure 12 came from a demonstrator who was instructed to execute the same turn many times. If this were the case, the different turning radii would then be attributed to the sub-optimal execution of the maneuver. The numerical level of expertise of the demonstrator could then be found by sweeping the value of α until more than one subgoal is consistently discovered – in this case $\alpha = 10$. Aside from serving as an indicator of expertise, this value of α could then be used as a starting point to interpret future demonstrations which may contain more than one subgoal.

2) Autonomous Execution of Learned Subgoals: Once subgoal rewards are learned, they can be converted to options as presented in Section V-F. Figure 13 shows a comparison of demonstrated maneuvers (from the original demonstration in Section III-F1) to the autonomous execution of the corresponding learned subgoal option for three of the six learned subgoals (colors correspond to Figure 10).

The autonomously executed maneuvers match the original demonstration closely, even though *no additional learning* was performed beyond the GPDP step in Algorithm 2. Also, each maneuver was demonstrated *once* in the single trajectory from Figure 10 from which the subgoals were learned and executed. These results demonstrate the ability of GPSRL to learn meaningful subgoals and execute corresponding autonomous policies from a single, unsegmented demonstration without user intervention or manual partitioning. The only necessary input to the algorithm (aside from parameter settings and the demonstration itself) is the state transition function described in Section V-A.

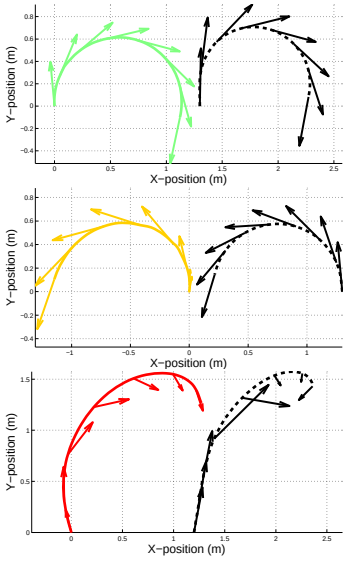


Fig. 13: Comparison of demonstrated maneuvers (from Figure 10, shown here in black) to the autonomous execution of the corresponding learned subgoal options (colored to correspond with the subgoals in Figure 10).

VII. Conclusions and Future Work

While reward learning from demonstration is a promising method of inferring a rich and transferable representation of the demonstrator's intents, current algorithms suffer from intractability and inefficiency in large domains due to the assumption that the demonstrator is maximizing a *single* reward function. To address this, a Bayesian nonparametric reward learning framework is developed that infers multiple reward functions from a single, unsegmented demonstration. The algorithm is shown to have both performance and computational advantages over existing methods. The framework is further extended to general continuous demonstration domains using Gaussian process reward representations, avoiding the need for discretization of the state space. Experimental results are given which demonstrate the ability of the new methods to learn challenging maneuvers from demonstration on a quadrotor helicopter and a remote-controlled car.

There are several extensions to the contributions presented in the paper which could serve as areas of future work. Since the computational complexity of the sampling procedure scales with the amount of demonstration data to be analyzed, another method for improving algorithmic efficiency could use the Gaussian process framework to represent trajectories instead of tabular storage of each observed state-action pair. This would require learning a Gaussian process which maps demonstration states to the observed actions taken. While such a representation would obviously be an approximation of the true observations, many GP sparsification methods exist [64, 65] which could greatly reduce the memory and computational requirements of the reward learning framework in cases where there is a large amount of demonstration data.

References

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

[2] M Lopes, F Melo, and L Montesano. Active Learning for Reward Estimation in Inverse Reinforcement Learning. *Machine Learning and Knowledge Discovery in Databases*, pages 31–46, 2009.

[3] T J Walsh, K Subramanian, M L Littman, and C Diuk. Generalizing Apprenticeship Learning across Hypothesis Classes. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1119–1126. Omnipress, 2010.

[4] J J Steil, F Röthling, R Haschke, and H Ritter. Situated robot learning for multi-modal instruction and imitation of grasping. *Robotics and Autonomous Systems*, 47(2-3):129–141, 2004.

[5] J Nakanishi. Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems*, 47(2-3):79–91, 2004.

[6] N Ratliff, D Bradley, and J A Bagnell. Boosting structured prediction for imitation learning. In *Advances in Neural Information Processing Systems 19*, 2007.

[7] J Z Kolter, P Abbeel, and A Y Ng. Hierarchical Apprenticeship Learning , with Application to Quadruped Locomotion. In J C Platt, D Koller, Y Singer, and S Roweis, editors, *Advances in Neural Information Processing Systems*, volume 1. MIT Press, 2008.

[8] C.G. Atkeson and S. Schaal. Robot learning from demonstration. In *Machine Learning International Workshop*, number 1994, pages 12–20. Citeseer, 1997.

[9] J Chen and A Zelinsky. Programing by Demonstration: Coping with Suboptimal Teaching Actions. *The International Journal of Robotics Research*, 22(5):299–319, 2003.

[10] A Chella. A posture sequence learning system for an anthropomorphic robotic hand. *Robotics and Autonomous Systems*, 47(2-3):143–152, 2004.

[11] P Abbeel and A Y Ng. Apprenticeship learning via inverse reinforcement learning. *Twentyfirst international conference on Machine learning ICML 04*, (ICML):1, 2004.

[12] D A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[13] W D Smart. *Making Reinforcement Learning Work on Real Robots*. PhD thesis, Department of Computer Science, Brown University, 2002.

[14] A Y Ng, A Coates, M Diel, V Ganapathi, J Schulte, B Tse, E Berger, and E Liang. Autonomous inverted helicopter flight via reinforcement learning. *International Symposium on Experimental Robotics*, 21:1–10, 2004.

[15] U Syed and R E Schapire. A Game-Theoretic Approach to Apprenticeship Learning. *Advances in Neural Information Processing Systems 20*, 20:1–8, 2008.

[16] R S Sutton, D Precup, and S Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 1999.

[17] G Konidaris, S Kuindersma, R Grupen, and A Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31(3):360–375, 2011.

[18] S Niekum, S Osentoski, G Konidaris, and A G Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[19] D Hewlett, T J Walsh, and P R Cohen. Teaching and executing verb phrases. In *Proceedings of the First Joint IEEE International Conference on Development and Learning and on Epigenetic Robotics (ICDL-Epirob)*, 2011.

[20] S Boyd, L E Ghaoui, E Feron, and V Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*, volume 15 of *SIAM Studies in Applied Mathematics*. SIAM, 1994.

[21] A Y Ng and S Russell. Algorithms for inverse reinforcement learning. In *Proc. of the 17th International Conference on Machine Learning*, pages 663–670, 2000.

[22] Christos Dimitrakakis and Constantin Rothkopf. Bayesian multitask inverse reinforcement learning. In *9th European Workshop on Reinforcement Learning (EWRL 2011)*, number EPFL-CONF-170268, 2011.

[23] N D Ratliff, J A Bagnell, and M A Zinkevich. Maximum margin planning. *Proc. of the 23rd International Conference on Machine Learning*, pages 729–736, 2006.

[24] B D Ziebart, A Maas, J A Bagnell, and A K Dey. Maximum Entropy Inverse Reinforcement Learning. In *Proc AAAI*, pages 1433–1438. AAAI Press, 2008.

[25] G Neu and C Szepesvari. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proc. UAI*, 2007.

[26] D Ramachandran and E Amir. Bayesian inverse reinforcement learning. *IJCAI*, 2007.

[27] D P Bertsekas and J N Tsitsiklis. *Neuro-Dynamic Programming*, volume 5 of *the Optimization and Neural Computation Series*. Athena Scientific, 1996.

[28] D H Grollman and O C Jenkins. Sparse incremental learning for interactive robot control policy estimation. *2008 IEEE Int. Conf. on Robotics and Automation*, 2008.

[29] E B Fox, E B Sudderth, M I Jordan, and A S Willsky. Nonparametric Bayesian Learning of Switching Linear Dynamical Systems. *Electrical Engineering*, 21(1), 2008.

[30] S Chiappa and J Peters. Movement extraction by detecting dynamics switches and repetitions. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[31] M Alvarez, J Peters, B Scholkopf, and N Lawrence. Switched Latent Force Models for Movement Segmentation. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[32] M. Toussaint, M. Gienger, and C. Goerick. Optimization of sequential attractor-based movement for compact behaviour generation. In *7th IEEE-RAS International Conference on Humanoid Robots*, pages 122–129, Nov 2007.

[33] Elmar A. Rückert, Gerhard Neumann, Marc Toussaint, and Wolfgang Maass. Learned graphical models for probabilistic planning provide a new class of movement primitives. *Frontiers in Computational Neuroscience*, 6(97), 2013.

[34] Hiroyuki Miyamoto, Stefan Schaal, Francesca Gandolfo, Hiroaki Gomi, Yasuharu Koike, Rieko Osu, Eri Nakano, Yasuhiro Wada, and Mitsuo Kawato. A kendama learning robot based on bi-directional theory. *Neural Networks*, 9(8):1281 – 1302, 1996. Four Major Hypotheses in Neuroscience.

[35] A McGovern and A G Barto. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. In *Machine Learning*, volume 85. Citeseer, 2001.

[36] I Menache, S Mannor, and N Shimkin. Q-Cut - Dynamic Discovery of Sub-Goals in Reinforcement Learning. *Proc. 13th European Conf. on Machine Learning*, 2002.

[37] O Simsek, A P Wolfe, and A G Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. *Proc. 22nd Int. Conf. on Machine Learning*, 2005.

[38] M Stolle and D Precup. Learning Options in Reinforcement Learning. *Abstraction Reformulation and Approximation*, 2371:212–223, 2002.

[39] O Simsek and A G Barto. Learning skills in reinforcement learning using relative novelty. *Abstraction Reformulation And Approximation Proceedings*, 3607:367–374, 2005.

[40] A G Barto, S J Bradtke, and S P Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138, 1995.

[41] B Michini and J P How. Bayesian Nonparametric Inverse Reinforcement Learning. In *European Conference on Machine Learning*, Bristol, United Kingdom, 2012.

[42] B. Michini, M. Cutler, and J.P. How. Scalable reward learning from demonstration. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 303–308, May 2013.

[43] R S Sutton and A G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[44] J Pitman. *Combinatorial Stochastic Processes*. Springer-Verlag, Berlin, 2006.

[45] A Gelman, J B Carlin, H S Stern, and D B Rubin. *Bayesian Data Analysis*, volume 2 of *Texts in statistical science*. Chapman & Hall/CRC, 2004.

[46] C E Rasmussen and C K Williams. *Gaussian processes in machine learning*. Lecture Notes in Computer Science. The MIT Press, 2006.

[47] M Babes, V N Marivate, K Subramanian, and M L Littman. Apprenticeship learning about multiple intentions. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 897–904, 2011.

[48] J Choi and K Kim. Nonparametric bayesian inverse reinforcement learning for multiple reward functions. In *Advances in Neural Information Processing Systems (NIPS) 25*, pages 314–322, 2012.

[49] C L Baker, R Saxe, and J B Tenenbaum. Action understanding as inverse planning. *Cognition*, 113(3):329–349, 2009.

[50] S Geman and D Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, 1984.

[51] M D Escobar and M West. Bayesian density estimation using mixtures. *Journal of the American Statistical Association*, 90(430):577– 588, 1995.

[52] R M Neal. Markov Chain Sampling Methods for Dirichlet Process Mixture Models. *Journal Of Computational And Graphical Statistics*, 9(2):249, 2000.

[53] C Andrieu, N De Freitas, A Doucet, and Michael I Jordan. An Introduction to MCMC for Machine Learning. *Science*, 50(1):5–43, 2003.

[54] J O Berger. *Statistical Decision Theory and Bayesian Analysis (Springer Series in Statistics)*. Springer, 1985.

[55] R M Neal. Probabilistic Inference Using Markov Chain Monte Carlo Methods. *Intelligence*, 62(September):144, 1993.

[56] G O Roberts and S K Sahu. Updating schemes, correlation structure, blocking and parameterisation for the Gibbs sampler. *Journal of the Royal Statistical Society - Series B: Statistical Methodology*, 59(2):291–317, 1997.

[57] P Abbeel. *Apprenticeship learning and reinforcement learning with application to robotic control*. PhD thesis, Stanford, 2008.

[58] Authors Authors. Bayesian nonparametric reward learning from demonstration. Technical report, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, July 2014.

[59] M Deisenroth, C E Rasmussen, and J Peters. Gaussian Process Dynamic Programming. *Neurocomputing*, 72(7-9):1508–1524, 2009.

[60] L Ljung. *System Identification*. John Wiley & Sons, Inc., 1999.

[61] J P How, B Bethke, A Frank, D Dale, and J Vian. Real-time indoor autonomous vehicle test environment, 2008.

[62] M Cutler and J P How. Actuator Constrained Trajectory Generation and Control for Variable-Pitch Quadrotors. In *Conference on Guidance, Navigation and Control*, 2012.

[63] E Velenis, E Frazzoli, and P Tsiotras. Steady-State Cornering Equilibria and Stabilization for a Vehicle During Extreme Operating Conditions. *International Journal Of Vehicle Autonomous Systems*, 8(2/3/4):217–241, 2010.

[64] L Csato and M Opper. Sparse Online Gaussian Processes. *Neural Computation*, 14(3):641–668, 2002.

[65] J Quinonero Candela and C E Rasmussen. A Unifying View of Sparse Approximate Gaussian Process Regression. *Journal of Machine Learning Research*, 6(3):1939–1959, 2005.