Scalable Reward Learning from Demonstration

Bernard Michini,★ Mark Cutler,★ and Jonathan P. How★★

Aerospace Controls Laboratory
Massachusetts Institute of Technology

Abstract— **Reward learning from demonstration is the task of inferring the intents or goals of an agent demonstrating a task. Inverse reinforcement learning methods utilize the Markov decision process (MDP) framework to learn rewards, but typically scale poorly since they rely on the calculation of optimal value functions. Several key modifications are made to a previously developed Bayesian nonparametric inverse reinforcement learning algorithm that avoid calculation of an optimal value function and no longer require discretization of the state or action spaces. Experimental results are given which demonstrate the ability of the resulting algorithm to scale to larger problems and learn in domains with continuous demonstrations.**

I. INTRODUCTION

Learning from demonstration is an intuitive method of teaching robotic and autonomous systems to perform potentially complex tasks or behaviors. While the literature contains many methods for learning from demonstration (see [1] for a survey), reward learning assumes a rational model of the demonstrator. The observations can then be used to invert the model and determine what reward structure the demonstrator was attempting to maximize. In essence, the *intents* or *goals* of the demonstrator are learned instead of just the policy being followed. Thus reward learning has the potential to provide a *robust* and *transferable* description of the task being demonstrated [2].

Inverse reinforcement learning (IRL) formalizes reward learning in the Markov decision process (MDP) framework [2]. IRL is the task of learning the reward function of an MDP given knowledge of the transition dynamics and a set of observed state-action pairs. Most IRL algorithms in the literature attempt to explain the observations using a single reward function defined over the entire state space [3]–[8]. However, this single reward function must be quite complex to explain most demonstrations of interest, and thus the reward function computation must be performed over a large (typically infinite) space of reward function candidates. In practice, this limits the applicability of IRL methods to small problems.

Previous work [9] developed a Bayesian nonparametric inverse reinforcement learning (BNIRL) algorithm to address the scalability of IRL methods to larger problems. The BNIRL algorithm automatically partitions the observed

demonstrations and finds a simple reward function to explain each partition using a Bayesian nonparametric mixture model. Using simple reward functions (which can be interpreted as *subgoals*) for each partition eliminates the need to search over a large candidate space. Also, the number of these partitions is assumed to be unconstrained and unknown a priori, allowing the algorithm to still explain complex behavior.

Results from [9] show that BNIRL performs similarly to a variety of conventional IRL methods for small problems, and furthermore can handle cyclic tasks which break the assumptions of the other methods. However, BNIRL (like other IRL methods) still relies on computing the optimal MDP value function in order to test reward function candidates. Calculating the optimal value function becomes infeasible for large state spaces [10], thus limiting the applicability of BNIRL to small problems.

The key contribution of this paper is to offer several effective methods to avoid computing the optimal MDP value function, enabling BNIRL to scale to much larger problem domains. In the first method, we modify BNIRL to use a framework known as Real-time Dynamic Programming (RTDP) [11]. RTDP effectively limits computation of the value function to necessary areas of the state space only. This allows 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 as in [9]. Experimental results are given for a Grid World domain and show order of magnitude speedups over IRL methods using exact solvers for large grid sizes.

In the second method, we modify BNIRL to utilize an existing closed-loop controller in 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. Experimental results are given for a quadrotor flight example which, if discretized, would require over 10^{10} states. In the experiment, quadrotor flight maneuvers are learned from a human demonstrator using only hand motions. The demonstration is recorded using a motion capture system and then analyzed by the BNIRL algorithm. Learned subgoal rewards (in the form of waypoints) are passed as commands to an autonomous quadrotor which executes the learned behavior in actual flight. The entire process from demonstration to reward learning to robotic execution takes on the order of 10 seconds to complete using a single computer. Thus, the results highlight

★Ph.D. Candidates, Aerospace Controls Laboratory, MIT, bmich[at]mit.edu and cutlerm[at]mit.edu
★★Richard C. Maclaurin Professor of Aeronautics and Astronautics, Aerospace Controls Laboratory, MIT, jhow[at]mit.edu

the ability of BNIRL to use data from a safe (and not necessarily dynamically feasible) demonstration environment and quickly learn subgoal rewards that can be used in the actual robotic system.

The paper is structured as follows. Section II presents preliminaries and summarizes the previously-developed BNIRL algorithm. Section III presents the two main contributions of the paper which allow for scalable reward learning. Experimental results are given in Section IV which demonstrate the ability of the improved algorithm to operate in large problem domains. Finally, Section V offers a discussion of the results and suggests areas of future work.

II. BACKGROUND

This section briefly covers some MDP preliminaries and summarizes the BNIRL algorithm from [9]. The reader is referred to the references for a more detailed discussion. 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

A finite-state Markov decision process is a tuple (S, A, T, γ, R) where S is a set of M states, A is a set of actions, $T : S \times A \times S \mapsto [0, 1]$ is the function of transition probabilities such that $T(s_1, a, s_2)$ is the probability of being in state s_2 after taking action a from state s_1, $R : S \mapsto \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. An MDP/R is defined as an MDP for which everything is specified except the state reward function $R(s)$.

A stationary policy is a function $\pi : S \mapsto A$. It is assumed that the expert executes a stationary (but potentially suboptimal) policy. From [10] we have the following set of 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 $V^\pi(s)$:

$$V^\pi(s, R) = E_\pi \left[\sum_{i=0}^{\infty} \gamma^i R(s_i) \middle| s_0 = s \right] \quad (1)$$

The value function satisfies the following Bellman Equation for all $s \in S$:

$$V^\pi(s, R) = R(s) + \gamma \left[\sum_{s'} T(s, \pi(s), s') V^\pi(s') \right] \quad (2)$$

The so-called Q-function (or action-value 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 for M iff, for all $s \in S$:

$$\pi(s) = \operatorname*{argmax}_{a \in A} Q^\pi(s, a, R) \quad (3)$$

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

B. Bayesian Nonparametric IRL

Inverse reinforcement learning (IRL) [2] is defined formally as the task of taking an MDP/R along with a set of N observed state-action pairs $\boldsymbol{O} = \{(s_1, a_1), (s_2, a_2), \dots, (s_N, a_N)\}$ and finding a reward function $\widehat{R}(s)$ whose corresponding optimal policy π^* matches the actions taken in $\boldsymbol{O}$.

The IRL problem is ill-posed since, for example, $\widehat{R}(s) = c \ \ \forall s \in S$ makes any set of state-action pairs $\boldsymbol{O}$ trivially optimal. Additionally, $\boldsymbol{O}$ may contain inconsistent information, i.e. (s_i, a_1) and (s_i, a_2) where $a_1 \neq a_2$. Existing IRL methods resolve the ambiguity by restricting the reward function to be of a certain form (e.g., linear-in-features) [3]–[5,7,8]. Moreover, most IRL methods attempt to explain the entire observations set $\boldsymbol{O}$ with a single, complex reward function. This places a large computational burden on the algorithm, since the space of candidate reward functions to be searched is extremely large and substantial computational effort is required to test each candidate. In practice, this has limited the application of IRL to small problems.

In BNIRL we avoid the costly search for a single complex reward function by partitioning the demonstration set and explaining each partition with a simple reward function. The simple reward function $R_g(s)$ consists of a positive reward at a single coordinate g in the state (or feature) space, and zero elsewhere, which can thus be interpreted as a "subgoal":

$$R_g(s) = \begin{cases} c & \text{at state } g \\ 0 & \text{at all other states} \end{cases} \quad (4)$$

The assumption is made that the candidate subgoals come from a finite set $g \in G$ which is constrained to those states (or features) observed in the demonstration, i.e.:

$$G = \{g \in S : g = s_i \text{ for some observation } O_i = (s_i, a_i)\} \quad (5)$$

In other words, it is assumed that the demonstrator achieves each of his subgoals at some point in the demonstration. This constrains the set of candidate reward functions to a finite set which scales with the size of the demonstration set $\boldsymbol{O}$ (as opposed to an infinite set defined over the entire state space in other IRL methods), enabling efficient reward inference.

To efficiently partition the demonstration set and assign subgoal reward functions to each partition, a Bayesian nonparametric generative model is assumed. A nonparametric model is chosen so that the number of partitions need not be constrained nor specified a priori. Each observation O_i is assigned to partition z_i with the associated subgoal reward function R_{z_i}. More specifically, the posterior probability of assignment z_i can be written as:

$$P(z_i | z_{-i}, \boldsymbol{O}) \propto \underbrace{P(z_i | z_{-i})}_{\text{CRP}} \ \underbrace{P(O_i | R_{z_i})}_{\text{likelihood}} \quad (6)$$

where a Chinese restaurant process (CRP) prior [12] is placed over partitions, and an action likelihood is defined for observations O_i given subgoal reward function R_{z_i}.

The CRP prior in (6) determines the probability of partition assignment z_i given all of the other current partition assignments z_{-i} as follows:

$$P(z_i = j|z_{-i}) = \begin{cases} \frac{\sum (z_{-i}=j)}{N-1+\eta} & \text{If partition } j \text{ already exists} \\ \frac{\eta}{N-1+\eta} & \text{If partition } j \text{ is new} \end{cases}$$
(7)

where N is the total number of observations in $\boldsymbol{O}$ and η is the *concentration parameter*. Intuitively, z_i is more likely to join large existing partitions, but there is always finite probability (governed by η) that z_i will start a new partition.

The action likelihood term in (6) encodes our model of rationality of the demonstrator. It determines the likelihood of observation $O_i = (s_i, a_i)$, given that the demonstrator is using subgoal reward function R_{z_i}. An exponential rationality model (similar to that in [6] and [13]) is used as follows:

$$P(O_i|R_{z_i}) = P(a_i|s_i, z_i) \propto e^{\alpha Q^*(s_i, a_i, R_{z_i})}$$
(8)

where the parameter α represents our degree of confidence in the demonstrator's ability to maximize reward. It is noted that in order to evaluate Q^* in (8), the optimal value function $V^*(s, a, R_{z_i})$ must be computed. Exact calculation of V^* becomes infeasible as the state space becomes larger (the so-called "curse of dimensionality" [10]), and is the main issue addressed in the following Section III.

It can be verified that the posterior distribution (6) is exchangeable (see [14]), i.e. that the order in which the observations $O_{1...N}$ arrive does not affect the joint posterior probability $P(\boldsymbol{z}|\boldsymbol{O})$. Thus sampling of the posterior can be done efficiently using Gibbs sampling [15], assuming that each observation is the last to arrive [16,17]. Each Gibbs sampling sweep proceeds as follows:

Algorithm 1 BNIRL Sampling Procedure

1: **for** each observation $O_i \in \boldsymbol{O}$ **do**
2:    **for** each current partition j **do**
3:       Calculate $p(z_i = j|z_{-i}, \boldsymbol{O})$ using (6)
4:    **end for**
5:    Calculate $p(z_i = k|z_{-i}, \boldsymbol{O})$ for a new partition k with associated subgoal reward R_k, drawn uniformly at random from the set G defined by (5)
6:    Normalize the set of probabilities calculated in lines 2–4 and 5, and sample partition assignment z_i
7: **end for**

Note that the number of partitions can be potentially infinite but is upper-bounded by the size of the observation set $\boldsymbol{O}$. Thus, for finite observation sets the sampling process is always feasible. The resulting samples for each z_i comprise an approximate posterior over subgoal assignments, the mode of which represents the most-probable set of demonstrator reward functions. The reader can refer to the previous BNIRL paper [9] for algorithmic details and convergence proofs.

III. ACTION LIKELIHOOD APPROXIMATION

The action likelihood (8) requires evaluation of the optimal action-value function $Q^*(s_i, a_i, R_{z_i})$ which, in turn, involves computing the optimal value function $V^*(s, R_{z_i})$.

This computation must be performed for each candidate reward function R (most other IRL methods [3]–[8] have the same requirement). It is well-known that the optimal value function requires substantial computational effort to compute, which for large state spaces becomes infeasible [18].

As a result, an approximation for the action likelihood (8) is required in order to scale reward learning to large problems. The following section describes two such approximations, and the experimental results in Section IV show that both scale well to large domains.

A. Real-time Dynamic Programming

One method of approximating the action likelihood (8) is to approximate the optimal action-value function Q^* itself. Approximating Q^* given a fully-specified MDP has been a popular area of research, known either as model-based reinforcement learning [10] or approximate dynamic programming [18].

Real-time dynamic programming (RTDP) [11] is one such method particularly well-suited to IRL. The basic premise of RTDP is to start with a set of sample states $\tilde{S}$, which is a small subset of the full state-space, i.e. $\tilde{S} \subset S$. Value iteration [10] is then performed on the sample states to generate a value function which is defined only over $\tilde{S}$. A greedy simulation is performed using $V^*(\tilde{S})$ starting from some state $s \in \tilde{S}$. Each state encountered in the simulation (as well as any neighboring states) is then added to $\tilde{S}$, and the process repeats.

As an example of using RTDP in BNIRL, consider the Grid World domain shown in Figure 1. 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, which indicate the action 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 1a. 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 $\tilde{S}$. The cycle repeats, and Figures 1b and 1c shows the progression of sample states and corresponding value functions. The process terminates when the greedy simulation fails to add any new states to $\tilde{S}$.

It is clear from the figures that RTDP only adds states necessary to improve the quality of the value function around $\boldsymbol{O}_i$, thus avoiding unnecessary computation in remote states. The net result is a reward learning algorithm that scales with the size of the demonstration set $\boldsymbol{O}$, and is agnostic to how large the surrounding state space may be. Experimental results in Section IV show that BNIRL combined with RTDP results in order of magnitude computation time decreases as compared to methods which use exact value function solvers.
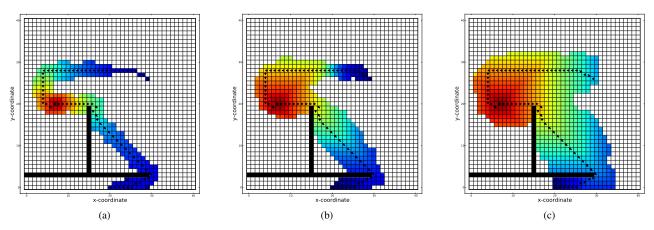
Fig. 1: Progression of real-time dynamic programming [11] sample states for the Grid World example. The algorithm starts with the initial set (a) based on the demonstration set (denoted by arrows), and uses greedy simulations to progressively expand the set of sample states (b and c) over which value iteration is performed.

B. Action Comparison

Many learning scenarios involve demonstrations in a continuous domain. Before Q^* can be calculated with conventional techniques, the domain must be discretized. Even for relatively simple domains, however, the discretization process can result in an extremely large state space.

Take, for instance, the 2-dimensional quadrotor model shown in Figure 2. The state-space is six-dimensional (two positions, one angle, and their time-derivatives). Even using modest discretization intervals (1cm, 1cm/s, $\pi/16$ rad, $\pi/16$ rad/sec) would require over 10^{10} states to cover a 1-meter by 1-meter grid. This is unwieldy even for approximate dynamic programming/model-based RL methods. Thus, trying to approximate Q^* for such domains quickly becomes infeasible.
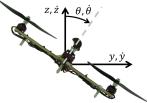

Fig. 2: Two-dimensional quadrotor model, showing y, z, and θ pose states along with $\dot{y}$, $\dot{z}$, and $\dot{\theta}$ velocity states.

An alternative to approximating Q^* is to instead approximate the entire action likelihood (8) itself. In words, (8) represents the likelihood that the demonstrator took action a_i from state s_i in order to maximize the subgoal reward R_{z_i}. As defined in (4), BNIRL subgoal rewards comprise a single positive reward for reaching some coordinate in the state (or feature) space. Thus, approximating (8) would simply require a measure of how likely action a_i would be if the demonstrator wanted to go from s_i to subgoal z_i.

One method for approximating this likelihood would be to compare action a_i with the action a_{CL} given by some closed-loop controller tasked with getting from s_i to z_i. An approximate action likelihood to use in place of (8) would

thus be:

$$P(O_i|R_{z_i}) = P(a_i|s_i, z_i) \propto e^{-\alpha||a_i - a_{CL}||_2} \qquad (9)$$

where a_{CL} is the action given by some closed-loop controller attempting to go from s_i to subgoal z_i. It is noted that the scaling of the norm $||a_i - a_{CL}||_2$ is inconsequential, since probabilities are normalized in Step 1c of the BNIRL sampling procedure (Algorithm 1).

The form of the closed-loop controller is problem-dependent, but in many cases a simple controller can be easily synthesized (or already exists). Take, for example, the 2-dimensional quadrotor in Figure 2. Let the states of the demonstration be a set of observed poses $s_i = (x_i, z_i, \theta_i)$ and the "actions" of the demonstration be the corresponding set of observed velocities $a_i = (\dot{x}_i, \dot{z}_i, \dot{\theta}_i)$. A simple controller would simply generate an action a_{CL} that commands a velocity in the direction of the pose error between s_i and z_i, i.e.:

$$a_{CL} \propto z_i - s_i \qquad (10)$$

While this may seem like an overly-simple control law, it is used to generate experimental results in Section IV which demonstrate the successful learning of autonomous quadrotor flight maneuvers from hand-held recorded demonstrations.

We see action comparison as a powerful method for approximating the likelihood (8) for several reasons. First, the method requires no discretization of the state or action spaces, as would be the case for methods which attempt to approximate Q^*. This makes the method well-suited for domains where the demonstration domain is continuous. Second, calculation of the control action a_{CL} is typically extremely fast compared to calculating (or approximating) an entire action-value function Q^*. This allows for real-time reward learning in many situations, as is shown in Section IV. Finally, the form of the closed-loop controller can be refined based on the degree of knowledge of the expert, enabling a trade-off between computational complexity and accuracy of the action likelihood approximation.

IV. EXPERIMENTS

The following section presents experimental results which apply the two action likelihood approximations described in Section IV to relatively large problem domains.

A. Grid World using RTDP

Consider the Grid World example presented in Section III-A (shown in Figure 1). 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 [3], a representative conventional IRL method) using full value iteration, BNIRL using RTDP, Abbeel IRL using RTDP, and BNIRL using parallelized RTDP. In the parallel BNIRL case, the pre-computation of the required approximate value functions is done in parallel on a cluster of 25 computing cores. The ability to compute value functions in parallel is a feature of the BNIRL algorithm (since the number of reward function candidates is finite and known a priori, see [9]). Abbeel IRL (as well as other conventional IRL methods [3]–[8]) 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, and results are only meant to demonstrate trends.

Figure 3a shows average CPU run-times of each method (lower is better) for Grid World domains ranging from 100 to 1,000,000 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 computing time). 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, the 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 RTPD. This is due to the fact that 25 computing cores can be utilized in parallel to calculate the necessary value functions for the BNIRL sampling procedure.

To ensure that the RTDP Q^* approximation does not affect the quality of the learned reward function, Figure 3b 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 than the full value iteration methods.

B. Learning Flight Maneuvers from Hand-Held Demonstrations

Note: Please see the accompanying video for further

explanation of the results in this section.

To test the action comparison likelihood approximation described in Section III-B, 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 4a) while the pose and velocities of the quadrotor are tracked and recorded by an indoor motion capture system down-sampled to 20Hz (Figure 4b). Using the 2-dimensional quadrotor model described in Section III-B and the closed-loop controller action comparison likelihood defined by (9) and (10), the BNIRL algorithm is used to generate an approximate posterior distribution over the demonstrator's subgoals. Figure 4c 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 [19] using the flight control law described in [20]. Figure 4d plots the hand-held trajectory against the autonomous flight, showing a close matchup between the demonstration and the resulting learned behavior.

To demonstrate the ability of BNIRL to handle cyclic, repetitive demonstrations, Figure 5 shows a cluttered trajectory where the demonstrator moves randomly between the four corners of a square. Overlayed are the four subgoals of the converged posterior, which correctly identify the four key subgoals inherent in the demonstration.
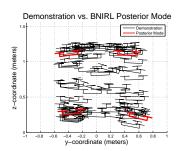

Fig. 5: A cluttered trajectory in which the demonstrator moves randomly between the four corners of a square is shown in black. The BNIRL posterior mode is shown in red, which consists of four subgoals, one at each corner of the square as expected.

Figure 6a shows another example, this time where the demonstrated trajectory is a flip. As shown in Figure 6b, 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. The subgoal waypoints are executed by the autonomous flight controller and the actual flight path is overlayed on Figure 6a, again showing the matchup between demonstrated and learned behavior.

Finally, it is noted that the BNIRL sampling process for the three examples above takes roughly three seconds to
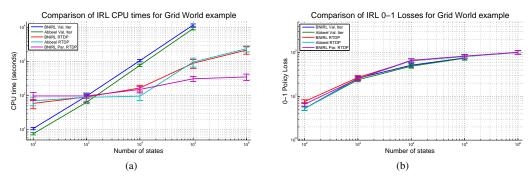
Fig. 3: Comparison of average CPU runtimes for various IRL algorithms for the Grid World example (left), along with the average corresponding 0-1 policy loss. Both plots use a log-log scale, and averages were take over 30 samples.
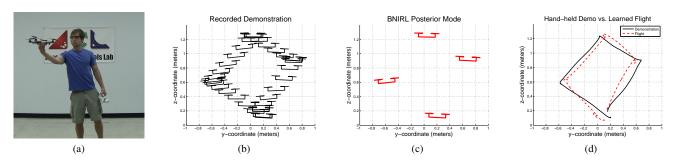

Fig. 4: A human demonstrator motions with a disabled quadrotor (a), while an indoor motion capture system records and downsamples demonstration (b). The BNIRL algorithm with action comparison likelihood converges to a mode posterior with four subgoals, one at each corner of the demonstrated trajectory (c). Finally, an autonomous quadrotor takes the subgoals as waypoints and executes the learned trajectory in actual flight (d).
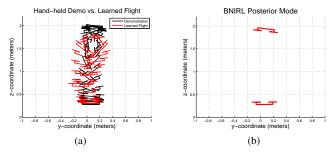

Fig. 6: A hand-held demonstrated quadrotor flip is shown in black (a). The BNIRL posterior mode in this case converges to two subgoals, one at the bottom and one (inverted) at the top of the flip trajectory (b). An autonomous quadrotor takes the subgoals as waypoints and executes the learned trajectory in actual flight, shown in red (a).

converge to the posterior mode. This is due to the fact that evaluation of the closed-loop control action in (9) is extremely fast, making BNIRL suitable for real-time reward learning.

V. CONCLUSIONS AND FUTURE WORK

This paper presents several improvements to the previously developed Bayesian nonparametric inverse reinforcement learning algorithm (BNIRL) [9], allowing for scalable and real-time reward learning from demonstration. The improvements center around approximating the action likelihood function, which previously required finding an optimal MDP action-value function Q^*.

The experimental results presented demonstrate several fundamental improvements over conventional IRL reward learning methods. BNIRL limits the size of the candidate reward space to a finite set, allowing for parallelized precomputation of (approximate) action value functions. The BNIRL likelihood function can be approximated using action comparison to an existing closed-loop controller, avoiding the need to discretize the state space and allowing for learning in continuous demonstration domains. Finally, the results demonstrate the ability of the algorithm to take data from a safe (and not necessarily dynamically feasible) demonstration domain and use it to learn meaningful subgoal rewards that can be executed by the actual robotic system.

There are two main areas of ongoing work. First, the application of BNIRL to larger continuous demonstration domains is being explored (to learn 3-dimensional flight maneuvers, for instance). This involves using a more complex closed-loop controller for action comparison (i.e. a full 3-dimensional flight controller) and testing the sensitivity of the resulting approximate posterior to such a controller. Second, the application of BNIRL to human decision-making domains is also being explored. Such domains typically have enormous state-action spaces and demonstrations which are comprised of much longer observation sequences, both of which pose many challenges to reward learning frameworks.

REFERENCES

[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

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

[3] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," *Twentyfirst international conference on Machine learning ICML 04*, no. ICML, p. 1, 2004.

[4] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum Entropy Inverse Reinforcement Learning," in *Proc AAAI*. AAAI Press, 2008, pp. 1433–1438.

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

[6] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning," *IJCAI*, pp. 2586–2591, 2007.

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

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

[9] B. Michini and J. P. How, "Bayesian Nonparametric Inverse Reinforcement Learning," in *European Conference on Machine Learning (to appear)*, 2012. [Online]. Available: http://acl.mit.edu/papers/michini-ecml-2012.pdf

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

[11] A. G. Barto, S. J. Bradtke, and S. P. Singh, "Learning to act using real-time dynamic programming," *Artificial Intelligence*, vol. 72, no. 1-2, pp. 81–138, 1995.

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

[13] C. L. Baker, R. Saxe, and J. B. Tenenbaum, "Action understanding as inverse planning." *Cognition*, vol. 113, no. 3, pp. 329–349, 2009.

[14] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin, *Bayesian Data Analysis*, ser. Texts in statistical science, C. Hall Crc, Ed. Chapman & Hall/CRC, 2004, vol. 2.

[15] S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 721–741, 1984.

[16] M. D. Escobar and M. West, "Bayesian density estimation using mixtures," *Journal of the American Statistical Association*, vol. 90, no. 430, pp. 577– 588, 1995.

[17] R. M. Neal, "Markov Chain Sampling Methods for Dirichlet Process Mixture Models," *Journal Of Computational And Graphical Statistics*, vol. 9, no. 2, p. 249, 2000.

[18] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, ser. the Optimization and Neural Computation Series. Athena Scientific, 1996, vol. 5, no. 6.

[19] J. P. How, B. Bethke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," pp. 51–64, 2008.

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