Title
Active Preference-Based Learning of Reward Functions

ISBN
9780992374730

Authors
Sadigh, Dorsa
Dragan, Anca
Sastry, Shankar
et al.

Publication Date
2017

DOI
10.15607/rss.2017.xiii.053

Peer reviewed

Active Preference-Based Learning of Reward Functions

Dorsa Sadigh, Anca D. Dragan, Shankar Sastry, and Sanjit A. Seshia
University of California, Berkeley, {dsadigh, anca, sastry, sseshia}@eecs.berkeley.edu

***Abstract*—Our goal is to efficiently learn reward functions encoding a human's preferences for how a dynamical system should act. There are two challenges with this. First, in many problems it is difficult for people to provide demonstrations of the desired system trajectory (like a high-DOF robot arm motion or an aggressive driving maneuver), or to even assign how much numerical reward an action or trajectory should get. We build on work in label ranking and propose to learn from *preferences* (or comparisons) instead: the person provides the system a relative preference between two trajectories. Second, the learned reward function strongly depends on what environments and trajectories were experienced during the training phase. We thus take an *active* learning approach, in which the system decides on what preference queries to make. A novel aspect of our work is the complexity and continuous nature of the queries: continuous trajectories of a dynamical system in environments with other moving agents (humans or robots). We contribute a method for actively *synthesizing* queries that satisfy the dynamics of the system. Further, we learn the reward function from a continuous hypothesis space by maximizing the volume removed from the hypothesis space by each query. We assign weights to the hypothesis space in the form of a log-concave distribution and provide a bound on the number of iterations required to converge. We show that our algorithm converges faster to the desired reward compared to approaches that are not active or that do not synthesize queries in an autonomous driving domain. We then run a user study to put our method to the test with real people.**

I. Introduction

Reward functions play a central role in specifying how dynamical systems should act: how an end-user wants their assistive robot arm to move, or how they want their autonomous car to drive. For many systems, end-users have difficulty providing demonstrations of what they want. For instance, they cannot coordinate 7 degrees of freedom (DOFs) at a time [2], and they can only show the car how *they* drive, not how they want *the car* to drive [5]. In such cases, another option is for the system to regress a reward function from labeled state-action pairs, but assigning precise numeric reward values to observed robot actions is also difficult.

In this paper, we propose a *preference-based* approach to learning desired reward functions in a dynamical system. Instead of asking for demonstrations, or for the value of the reward function for a sample trajectory (e.g., "rate the safety of this driving maneuver from 1 to 10"), we ask people for their relative preference between two sample trajectories (e.g., "is ξ_1 more safe or less safe than ξ_2?").

Active preference-based learning has been successfully used in many domains [1, 6, 7, 14], but what makes applying it to learning reward functions difficult is the complexity of the queries, as well as the continuous nature of the underlying hypothesis space of possible reward functions. We focus on dynamical systems with continuous or hybrid discrete-continuous state. In this setting, queries consist of two candidate continuous state and action space trajectories that satisfy the system's dynamics, in an environment or scenario that the learning algorithm also needs to decide on, consisting of an initial state and the trajectories of *other* agents in the scene. Consider again the example of autonomous driving. In this situation, a query would consist of two trajectories for the car from some starting state among other cars following their own trajectories.

Typically in preference-based learning, the queries are actively selected by searching some discrete or sampled set (e.g., [3, 9, 12, 13, 20, 21]). Our first hypothesis is that in our setting, the continuous and high-dimensional nature of the queries renders relying on a discrete set ineffective. Preference-based work for such spaces has thus far collected data passively [18, 22]. Our second hypothesis is that active generation of queries leads to better reward functions faster.

We contribute an algorithm for *actively synthesizing* queries from scratch. We do continuous optimization in query space to maximize the expected volume removed from the hypothesis space. We use the human's response to assign weights to the hypothesis space in the form of a *log-concave distribution*, which provides an approximation of the objective via a Metropolis algorithm that makes it differentiable w.r.t. the query parameters. We provide a bound on the number of iterations required to converge.

We compare our algorithm to non-active and non-synthesis approaches to test our hypotheses. We use an experimental setup motivated by autonomous driving, and show that our approach converges faster to a desired reward function. Finally, we illustrate the performance of our algorithm in terms of accuracy of the reward function learned through an in-lab usability study.

II. Problem Statement

Modeling Choices. Our goal is to model the behavior and preferences of a human for how a dynamical system should act. We denote this dynamical system that should

match human preferences by $\mathcal{H}$, and it interacts with other systems (robots) in an environment. We model the overall system including $\mathcal{H}$, and all the other agents as a fully-observable dynamical system. The continuous state of the environment $x \in X$ includes the state of $\mathcal{H}$ and all the other agents. We let u_H denote the continuous control input of $\mathcal{H}$. For simplicity, we assume that there is only one other agent (robot) $\mathcal{R}$ locally visible in the environment, and we let u_R be its continuous control input. The dynamics of the state changes based on the actions of both agents through f_{HR}:

$$x^{t+1} = f_{HR}(x^t, u_R, u_H). \tag{1}$$

We define a trajectory $\xi \in \Xi$, where $\xi = (x^0, u_R^0, u_H^0), \ldots, (x^N, u_R^N, u_H^N)$ is a finite horizon sequence of states and actions of all agents. Here, Ξ is a set of all feasible continuous trajectories. A feasible trajectory is one that satisfies the dynamics of $\mathcal{H}$ and $\mathcal{R}$. We first parameterize the preference reward function as a linear combination of a set of features:

$$r_H(x^t, u_R^t, u_H^t) = \mathbf{w}^\top \phi(x^t, u_R^t, u_H^t), \tag{2}$$

where $\mathbf{w}$ is a vector of weights for the feature function $\phi(x^t, u_R^t, u_H^t)$ evaluated at every state and action pair. We assume a d-dimensional feature function, so $\phi(x^t, u_R^t, u_H^t) \in \mathbb{R}^d$. Further, for a finite horizon N, we let $\mathbf{x} = (x^0, \ldots, x^N)^\top$ be a sequence of states, $\mathbf{u}_H = (u_H^0, \ldots, u_H^N)^\top$ a sequence of human actions, and $\mathbf{u}_R = (u_R^0, \ldots, u_R^N)^\top$ a sequence of robot actions. We define R_H to be the human's expected reward over horizon N:

$$R_H(x^0, \mathbf{u}_R, \mathbf{u}_H) = \sum_{t=0}^{N} r_H(x^t, u^t, u_H^t). \tag{3}$$

For simpler notation, we combine the $N+1$ elements of ϕ so $\Phi = \sum_{t=0}^{N} \phi(x^t, u_R^t, u_H^t)$. Therefore, $\Phi(\xi)$ evaluates over a trajectory ξ. We finally reformulate the reward function as an inner product:

$$R_H(\xi) = \mathbf{w} \cdot \Phi(\xi). \tag{4}$$

Our goal is to learn R_H.

Approach Overview. Inverse Reinforcement Learning (IRL) [15, 19, 23] enables us to learn R_H through demonstrated trajectories. However, IRL requires the human to show demonstrations of the optimal sequence of actions. Providing demonstrations can be challenging for many human-robot tasks. Furthermore, generating interesting training data that actively explores R_H through atypical, interesting environments (which is necessary in many cases for resolving ambiguities) works well [8, 16] but in practice can make (good) demonstrations infeasible: the algorithm cannot physically manufacture environments, and therefore relies on simulation, which makes demonstrations only possible through teleoperation.

For these reasons, we assume demonstrations are not available in our work. Instead, we propose to leverage preference-based learning, which queries $\mathcal{H}$ to provide comparisons between two candidate trajectories. We propose a new approach for active learning of a reward function for human preferences through comparisons.

We split trajectories ξ into two parts: a scenario and the trajectory of the agent $\mathcal{H}$ whose reward function we are learning. We formalize a scenario to be the initial state of the environment as well as the sequence of actions for the other agent(s) $\mathcal{R}$, $\tau = (x^0, u_R^0, \ldots, u_R^N)$. Given an environment specified by scenario $\tau \in \mathcal{T}$, the human agent will provide a finite sequence of actions $\mathbf{u}_H = u_H^0, \ldots, u_H^N$ in response to scenario τ. This response, along with the scenario τ defines a trajectory ξ showing the evolution of the two systems together in the environment.

We iteratively synthesize queries where we ask the human to compare between two trajectories ξ_A and ξ_B defined over the same fixed scenario τ as shown in Fig. 1 (a). Their answer provides us information about $\mathbf{w}$. In what follows, we discuss how to update our probability distribution over $\mathbf{w}$ given the answer to a query, and how to actively synthesize queries in order to efficiently converge to the right $\mathbf{w}$.

III. Learning Reward Weights from Preferences of Synthesized Queries

In this section, we describe how we update a distribution over reward parameters (weights $\mathbf{w}$ in equation (4)) based on the answer of one query. We first assume that we are at iteration t of the algorithm and an already synthesized pair of trajectories ξ_A and ξ_B in a common scenario is given (we discuss in the next section how to generate such a query). We also assume $\mathcal{H}$ has provided her preference for this specific pair of trajectories at iteration t. Let her answer be I, with $I_t = +1$ if she prefers the former, and $I_t = -1$ if she prefers the latter. This answer gives us information about $\mathbf{w}$: she is more likely to say $+1$ if ξ_A has higher reward than ξ_B, and vice-versa, but she might not be exact in determining I_t. We thus model the probability $p(I|\mathbf{w})$ as noisily capturing the preference w.r.t. R_H:

$$p(I_t|\mathbf{w}) = \begin{cases} \frac{\exp(R_H(\xi_A))}{\exp(R_H(\xi_A))+\exp(R_H(\xi_B))} & I_t = +1 \\ \\ \frac{\exp(R_H(\xi_B))}{\exp(R_H(\xi_A))+\exp(R_H(\xi_B))} & I_t = -1 \end{cases} \tag{5}$$

We start with a prior over the space of all $\mathbf{w}$, i.e., $\mathbf{w}$ is uniformly distributed on the unit ball. Note that the scale of $\mathbf{w}$ does not change the preference I_t, so we can constrain $||\mathbf{w}|| \leq 1$ to lie in this unit ball. After receiving the input of the human I_t, we propose using a Bayesian update to find the new distribution of $\mathbf{w}$:

$$p(\mathbf{w}|I_t) \propto p(\mathbf{w}) \cdot p(I_t|\mathbf{w}). \tag{6}$$

Let

$$\varphi = \Phi(\xi_A) - \Phi(\xi_B) \tag{7}$$

Then our update function that multiplies the current distribution at every step is:

$$f_\varphi(\mathbf{w}) = p(I_t|\mathbf{w}) = \frac{1}{1+\exp(-I_t\mathbf{w}^\top\varphi)} \tag{8}$$

Updating the distribution of $\mathbf{w}$ allows us to reduce the probability of the undesired parts of the space of $\mathbf{w}$, and maintain the current probability of the preferred regions.

IV. Synthesizing Queries through Active Volume Removal

The previous section showed how to update the distribution over $\mathbf{w}$ after getting the response to a query. Here, we show how to synthesize the query in the first place: we want to find the next query such that it will help us remove as much volume (the integral of the unnormalized pdf over $\mathbf{w}$) as possible from the space of possible rewards.

Formulating Query Selection as Constrained Optimization. We synthesize experiments by maximizing the volume removed under the feasibility constraints of φ:

$$\begin{aligned} \max_{\varphi} \quad & \min\{\mathbb{E}[1-f_\varphi(\mathbf{w})], \mathbb{E}[1-f_{-\varphi}(\mathbf{w})]\} \\ \text{subject to} \quad & \varphi \in \mathbb{F} \end{aligned} \tag{9}$$

The constraint in this optimization requires φ to be in the feasible set $\mathbb{F}$:

$$\begin{aligned} \mathbb{F} = \{\varphi : \varphi = \Phi(\xi_A) - \Phi(\xi_B), \xi_A, \xi_B \in \Xi, \\ \tau = (x^0, \mathbf{u}_R^A) = (x^0, \mathbf{u}_R^B)\} \end{aligned} \tag{10}$$

which is a set of the difference of features over feasible trajectories $\xi_A, \xi_B \in \Xi$ defined over the same scenario τ.

In this optimization, we maximize the minimum of the split between the two spaces preferred by either choices of the human agent. Each term in the minimum is the volume removed depending on the input of the human I_t, i.e., the preference. The expectation is taken w.r.t. to distribution over $\mathbf{w}$. **Solution.** We first reformulate our problem as an unconstrained optimization, where we enforce the feasibility of trajectories by directly optimizing over the query components, $x^0, \mathbf{u}_R, \mathbf{u}_H^A, \mathbf{u}_H^B$, as opposed to optimizing in the desired feature difference φ:

$$\max_{x^0, \mathbf{u}_R, \mathbf{u}_H^A, \mathbf{u}_H^B} \quad \min\{\mathbb{E}[1-f_\varphi(\mathbf{w})], \mathbb{E}[1-f_{-\varphi}(\mathbf{w})]\} \tag{11}$$

Here, φ remains a function of $x^0, \mathbf{u}_R, \mathbf{u}_H^A, \mathbf{u}_H^B$. We will solve this by local optimization using a Quasi-Newton method (L-BFGS [4]). To do so, we need a differentiable objective.

The distribution $p(\mathbf{w})$ can become very complex and there is no simple way to compute the volume removed by a Bayesian update, let alone differentiate through it. We therefore resort to sampling, and optimize an approximation of the objective obtained via samples, where weights are sampled in proportion to their probability. Assume we sample $\mathbf{w}_1, \ldots, \mathbf{w}_M$ independently from the distribution $p(\mathbf{w})$. Then we can approximate $p(\mathbf{w})$ by the empirical distribution composed of point masses at $\mathbf{w}_i$'s:

$$p(\mathbf{w}) \sim \frac{1}{M}\sum_{i=1}^{M}\delta(\mathbf{w}_i). \tag{12}$$

Then the volume removed by an update $f_\varphi(\mathbf{w})$ can be approximated by:

$$\mathbb{E}[1-f_\varphi(\mathbf{w})] \simeq \frac{1}{M}\sum_{i=1}^{M}(1-f_\varphi(\mathbf{w}_i)). \tag{13}$$

Given such samples, the objective is now differentiable w.r.t. φ, which is differentiable w.r.t. the starting state and controls – the ingredients of the query which are the variables in (11). What remains is to get the actual samples. To do so, we take advantage of the fact that $p(\mathbf{w})$ is a log-concave function, and the update function $f_\varphi(\mathbf{w})$ defined here is also log-concave in $\mathbf{w}$ as shown in Fig. 1 (c); therefore, the posterior distribution of $\mathbf{w}$ stays log-concave. Note we do not need to renormalize the distribution $p(\mathbf{w})$ after a Bayesian update, i.e. divide $p(\mathbf{w})$ by its integral. Instead, we use Metropolis Markov Chain methods to sample from $p(\mathbf{w})$ without normalization.

Log-concavity is useful because we can take advantage of efficient polynomial time algorithms for sampling from the current $p(\mathbf{w})$ [17][1]. In practice, we use an adaptive Metropolis algorithm, where we initialize with a warm start by computing the mode of the distribution, and perform a Markov walk [11].

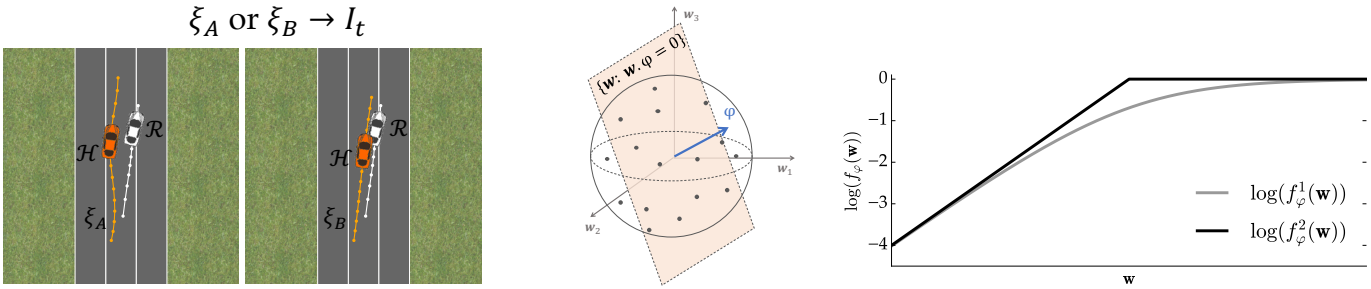
We could find the mode of f_φ from (8), but it requires a convex optimization. We instead speed this up by choosing a similar log-concave function whose mode evaluates to zero always, and which reduces the probability of undesired $\mathbf{w}$ by a factor of $\exp(I_t\mathbf{w}^\top\varphi)$:

$$f_\varphi(\mathbf{w}) = \min(1, \exp(I_t\mathbf{w}^\top\varphi)) \tag{14}$$

Fig. 1 (c) shows this simpler choice of update function in black with respect to $p(I_t|\mathbf{w})$ in gray. The shape is similar, but enables us to start from zero, and a Markov walk will efficiently sample from the space of $\mathbf{w}$.

In Fig. 1 (b), we show a simple diagram demonstrating our approach. Here, $\mathbf{w}$ first uniformly lies on a unit d-dimensional ball. For simplicity, here we show a 3D ball. The result of a query at every step is a state and trajectories that result in a feature difference vector φ, normal to the hyperplane $\{\mathbf{w} : \mathbf{w}^\top\varphi = 0\}$, whose direction represents the preference of the human, and its value lies in $\mathbb{F}$ defined in equation (10). We reduce the volume of $\mathbf{w}$ by reducing the probability distribution of the samples of $\mathbf{w}$ on the rejected side of the hyperplane through the update function $f_\varphi(\mathbf{w})$ that takes into account noise in the comparison.

[1]Note that computing the actual volume removed can be converted to the task of integrating a log-concave function, for which efficient algorithms do exist. The purpose of using samples instead is to have an expression suitable for maximization of volume removed, i.e. an expression differentiable w.r.t. the query.



(a) Preference query. (b) Query response effect. (c) log-concave update of $\mathbf{w}$.

Fig. 1: In (a) two candidate trajectories are provided for comparison to the human oracle. ξ_A shows a smoother trajectory without any collisions. In (b) the unit ball is representing the space of $\mathbf{w}$. We synthesize experiments that correspond to a separating hyperplane $\{\mathbf{w} : \mathbf{w} \cdot \varphi = 0\}$, and reweigh samples of $\mathbf{w}$ on each side of the hyperplane in order to update the distribution of $\mathbf{w}$. In (c) we show the two choices of log-concave update functions. Here, $f^1_\varphi(\mathbf{w}) = p(I_t|\mathbf{w})$ is the Bayesian update, and $f^2_\varphi(\mathbf{w}) = \min(1, \exp(I_t \mathbf{w}^\top \varphi))$ is our choice of simpler update function.

V. Algorithm and Performance Guarantees

Armed with a way of generating queries and a way of updating the reward distribution based on the human's answer to each query, we now present our method for actively learning a reward function in Alg. 1. The inputs to the algorithm are a set of features ϕ, the desired horizon N, the dynamics of the system f_{HR}, and the number of iterations *iter*. The goal is to converge to the true distribution of $\mathbf{w}$, which is equivalent to finding the preference reward function $R_H(\xi) = \mathbf{w} \cdot \Phi(\xi)$.

We first initialize this distribution to be uniform over a unit ball in line 3. Then, for M iterations, the algorithm repeats these steps: volume estimation, synthesizing a feasible query, querying the human, and updating the distribution.

We sample the space of $\mathbf{w}$ in line 5. Using these samples, and the dynamics of the system, SynthExps solves the optimization in equation (11): it synthesizes a feasible pair of trajectory that maximizes the expected volume removed from the distribution of $p(\mathbf{w})$. The answer to the query is received in line 7. We compute $f_\varphi(\mathbf{w})$, and update the distribution in line 10.

Algorithm 1 Preference-Based Learning of Reward Functions

1: **Input:** Features ϕ, horizon N, dynamics f, *iter*
2: **Output:** Distribution of $\mathbf{w}$: $p(\mathbf{w})$
3: Initialize $p(\mathbf{w}) \sim \text{Uniform}(B)$, for a unit ball B
4: **While** $t <$ *iter*:
5: $\quad W \leftarrow M$ samples from AdaptiveMetropolis($p(\mathbf{w})$)
6: $\quad (x^0, \mathbf{u}_R, \mathbf{u}^A_H, \mathbf{u}^B_H) \leftarrow \text{SynthExps}(W, f)$
7: $\quad I_t \leftarrow \text{QueryHuman}(x^0, \mathbf{u}_R, \mathbf{u}^A_H, \mathbf{u}^B_H)$
8: $\quad \varphi = \Phi(x^0, \mathbf{u}_R, \mathbf{u}^A_H) - \Phi(x^0, \mathbf{u}_R, \mathbf{u}^B_H)$
9: $\quad f_\varphi(\mathbf{w}) = \min(1, I_t \exp(\mathbf{w}^\top \varphi))$
10: $\quad p(\mathbf{w}) \leftarrow p(\mathbf{w}) \cdot f_\varphi(\mathbf{w})$
11: $\quad t \leftarrow t + 1$
12: **End for**

Regarding the convergence of Alg. 1, one cannot generally make any strict claims for several reasons: We replace the distribution $p(\mathbf{w})$ by an empirical distribution, which could introduce errors. The maximization in line 6 is via non-convex optimization which does not necessarily find the optimum, and even if it was guaranteed that the global optimum is found, it could potentially make very little progress in terms of volume removal, since the set $\mathbb{F}$ can be arbitrary.

Putting aside the issues of sampling and global optimization, we can compare what Alg. 1 does to the best one could do with the set $\mathbb{F}$. Alg. 1 can be thought of as a *greedy* volume removal.

Theorem V.1. *Under the following assumptions:*

- *The update function is f_φ as defined in equation* (8),
- *The human inputs are noisy similar to equation* (5),
- *The errors introduced by sampling and non-convex optimization are ignored,*

Alg. 1 removes at least $1 - \epsilon$ times as much volume as removed by the best adaptive strategy after $\ln(\frac{1}{\epsilon})$ times as many iterations.

Proof: Removed volume can be seen to be an adaptive submodular function, defined in terms of the choices φ and the human input I_t, as defined in [10]. It is also adaptive monotone; thus, the results of [10] imply that greedy volume removal for l steps in expectation removes at least $(1 - \exp(-l/k))\text{OPT}_k$ where OPT_k is the best solution any adaptive strategy can achieve after k steps. Setting $l = k \ln(\frac{1}{\epsilon})$ gives us the desired result.

One caveat is that in equation (8) the human input I_t is treated as worst case, i.e., in the synthesis step, one maximizes the *minimum* removed volume (over the possible I_t). Namely maximizing the following quantity:

$$\min\{\mathbb{E}_\mathbf{w}[1 - \Pr[I_t = +1|\mathbf{w}]]), \mathbb{E}_\mathbf{w}[1 - \Pr[I_t = -1|\mathbf{w}]])\}. \tag{15}$$

Normally the greedy strategy in adaptive submodular

maximization should treat I_t as probabilistic. In other words instead of the minimum one should typically maximize the following quantity:

$$\begin{aligned}\Pr[I_t = +1] \cdot \mathbb{E}_{\mathbf{w}}[1 - \Pr[I_t = +1|\mathbf{w}]] + \\ \Pr[I_t = -1] \cdot \mathbb{E}_{\mathbf{w}}[1 - \Pr[I_t = -1|\mathbf{w}]].\end{aligned} \tag{16}$$

However, note $\mathbb{E}_{\mathbf{w}}[1 - \Pr[I_t = +1|\mathbf{w}]]$ is simply $\Pr[I_t = -1]$ and similarly $\mathbb{E}_{\mathbf{w}}[1 - \Pr[I_t = +1|\mathbf{w}]]$ is $\Pr[I_t = +1]$. Therefore Alg. 1 is maximizing $\min(\Pr[I_t = -1], \Pr[I_t = +1])$, whereas greedy submodular maximization should be maximizing $2 \Pr[I_t = -1] \Pr[I_t = +1]$. It is easy to see that these two maximizations are equivalent, since the sum $\Pr[I_t = -1] + \Pr[I_t = +1] = 1$ is fixed. ■

VI. Simulation Experiment

In this section, we evaluate our theory using a semiautonomous driving example. Our goal is to learn people's preferred reward function for driving. In this context, our approach being preference-based is useful since it allows the users to only compare two candidate trajectories in various scenarios instead of requiring the user to demonstrate a full trajectory (of how they would like to drive, not how they actually drive). In addition, our approach being *active* enables choosing informative test cases that are otherwise difficult to encounter in driving scenarios. For example, we can address the preference of drivers in moral dilemma situations (deciding between two undesirable outcomes), which are very unlikely to arise in standard collected driving data. Finally, our approach *synthesizes* these test cases from scratch, which should help better exploit the continuous and high-dimensional space of queries. We put these advantages to the test in what follows.

Experimental Setup. We assume a human driven vehicle $\mathcal{H}$ living in an environment with another vehicle $\mathcal{R}$, and we synthesize trajectories containing two candidate sequence of actions for the human driven car, while for every comparison we fix the synthesized scenario (i.e., the initial state of the environment and the sequence of actions of $\mathcal{R}$). Fig. 1 (a) shows an example of this comparison. The white vehicle is $\mathcal{R}$, and the orange vehicle corresponds to $\mathcal{H}$. The white and orange lines show the path taken by the human and robot, respectively, in the two cases over a horizon of $N = 5$. We assume a simple point-mass dynamics model for both of the vehicles:

$$[\dot{x}\ \dot{y}\ \dot{\theta}\ \dot{v}] = \begin{bmatrix} v \cdot \cos(\theta) & v \cdot \sin(\theta) & v \cdot u_1 & u_2 - \alpha \cdot v \end{bmatrix}. \tag{17}$$

Here, the state $\mathbf{x} = \begin{bmatrix} x & y & \theta & v \end{bmatrix}$ includes the coordinates of the robot x and y, the heading θ, and the velocity v. The control input of this dynamical system is $\mathbf{u} = \begin{bmatrix} u_1 & u_2 \end{bmatrix}$, where u_1 is the steering input, and u_2 is the acceleration. We also use α as a friction coefficient.

We learn the reward function of the human's preferences based on queries similar to Fig. 1 (a). We define a set of features that allow representing this cost function. First, $f_1 \propto c_1 \cdot \exp(-c_2 \cdot d^2)$ corresponds to penalizing getting close to the boundaries of the road, where d is the distance between the vehicle and these boundaries, and c_1 and c_2 are appropriate scaling factors. We use a similar feature f_2 for enforcing staying within a single lane by penalizing leaving the boundaries of the lane. We also encourage higher speed for moving forward through $f_3 = (v - v_{\max})^2$, where v is the velocity of the vehicle, and $v_{\max}$ is the speed limit. Also, we would like the vehicle to have a heading along with the road using a feature $f_4 = \theta_H \cdot \vec{n}$, where θ_H is the heading of $\mathcal{H}$, and $\vec{n}$ is a normal vector along the road. Our last feature f_5 corresponds to collision avoidance, and is a non-spherical Gaussian over the distance of $\mathcal{H}$ and $\mathcal{R}$, whose major axis is along the robot's heading. Then, we aim to learn a distribution over the weights corresponding to these features $\mathbf{w} = [w_1, w_2, w_3, w_4, w_5]$ so $R_H = \mathbf{w} \cdot \Phi(\xi)$ best represents the preference reward function.

Conditions. We compare our algorithm with two baselines: non-active and non-synthesis.

First, we compare it to a *non-active* version. Instead of actively generating queries that will remove the most volume, we uniformly sample a scenario. We do not use totally random trajectories for the cars as this would be a weak baseline, instead we sample two candidate weight vectors $\mathbf{w}^A$ and $\mathbf{w}^B$ from the current distribution on $\mathbf{w}$. We then maximize the reward functions $\mathbf{w}^A \cdot \Phi(\xi_A)$ and $\mathbf{w}^B \cdot \Phi(\xi_B)$ to solve for optimal $\mathbf{u}_H^A$ and $\mathbf{u}_H^B$. That creates our query, comparing two reward functions, but no longer optimized to efficiently learn $\mathbf{w}$. The trajectories generated through this other approach are then used to query the human, and update the distribution of $p(\mathbf{w})$ similar to Alg.1. Second, we compare it against a *non-synthesis* (discrete) version. Instead of solving a continuous optimization problem to generate a query, we do a discrete search over a sampled set of queries.

Metric. We evaluate our results using a hidden reward function $R_{\text{true}} = \mathbf{w}_{\text{true}} \cdot \Phi(\xi)$. We query an ideal user who knows R_{true} and uses it to compare pairs of trajectories, and show the $\mathbf{w}$ computed by our algorithm efficiently converges to $\mathbf{w}_{\text{true}}$.

At every step of the iteration, we compute the following measure of convergence:

$$m = \mathbb{E}[\frac{\mathbf{w} \cdot \mathbf{w}_{\text{true}}}{|\mathbf{w}||\mathbf{w}_{\text{true}}|}]. \tag{18}$$

Here, m computes the average heading of the current distribution of $\mathbf{w}$ with respect to $\mathbf{w}_{\text{true}}$ – how similar the learned reward is. Since the prior distribution of $\mathbf{w}$ is symmetric (uniformly distributed on a unit ball), this expectation starts at 0, and moves closer to 1 at every step of the iteration.

Hypotheses.

H1. *The reward function learned through our algorithm is closer to the true reward compared to the non-active baseline.*

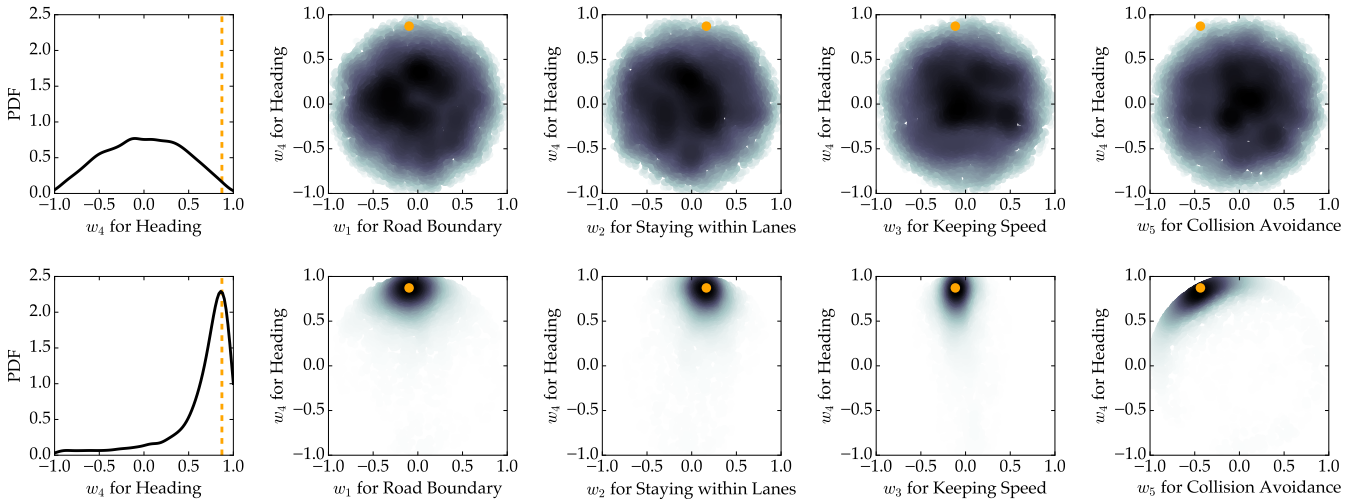


Fig. 2: Distribution of w_4, the weight for the heading feature, relative to the other features. The top plots shows the starting distribution, and the bottom plot shows the distribution at convergence. The orange dot and dotted line show the ground truth for the weights.

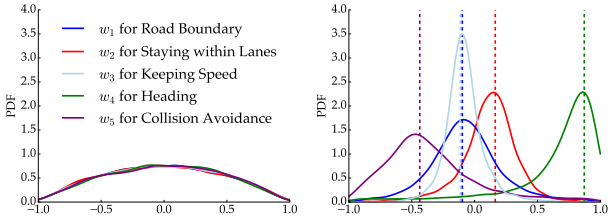


Fig. 3: Distribution over all weights before/after convergence. The dotted lines show the ground truth of the weights.

H2. *The reward function learned through our algorithm is closer to the true reward compared to the non-synthesis baseline.*

Results. We run a paired t-test for each hypothesis. Supporting H1, we find that our algorithm significantly outperforms the non-active version ($t(1999) = 122.379$, $p < .0001$), suggesting that it is important to be doing active learning in these continuous and high-dimensional query spaces. Supporting H2, we find that our algorithm significantly outperforms the non-synthesis version ($t(1999) = 35.39$, $p < .0001$), suggesting the importance of synthesizing queries instead of relying on a discrete set. Note these results hold even with conservative Bonferroni corrections for multiple comparisons.

Fig. 3 shows the distribution of $\mathbf{w}$ for our algorithm corresponding to the five features after 200 iterations, showing convergence to close to the true weights (in dotted lines). The mode of the distribution of w_1 has a negative weight enforcing staying within the roads, and w_2 has a positive weight enforcing staying within your own lane. w_3 also shows a slight preference for keeping speed, and w_4 shows a significant preference for keeping heading. w_5 also shows the distribution for the weight of collision avoidance.

Further, we show the initial and final two dimensional projection of the density of w_4, the weight of the heading feature, with respect to all the other $\mathbf{w}$'s in Fig. 2. This shift and convergence of the distribution is clear from the top plot to the bottom plot. The orange dot in all plots, as well as the orange dotted line show the ground truth of $\mathbf{w}_{\text{true}}$. Fig. 2 shows our algorithm clearly converges to close to the ground truth for all features.

We show the convergence of the distribution $p(\mathbf{w})$ in Fig 4. The dark red line shows the mean result of Alg. 1. It outperforms, according to metric m, both the non-active condition (black), as well as the non-synthesis (discretized) condition (dark blue). The results are the average of 10 runs of the algorithms (since sampling is nondeterministc). The bar graph in the middle also shows the converged value of m after 200 queries.

In a thought experiment, we also investigated to what extent we see these effects because we are generating real trajectories for dynamical systems. To test this, we used the three algorithms to produce abstract queries that a simulation could answer, consisting of only a feature difference φ as opposed to two feasible real trajectories. The value of m after 200 iterations for these versions of the algorithms are shown in the bar graph with lighter colors. Because these approaches are unencumbered by the need for feasibility of the queries (or even producing actual concrete queries), they overall perform better than when the algorithms need to generate real queries. In this case, being active is still important, but the ability

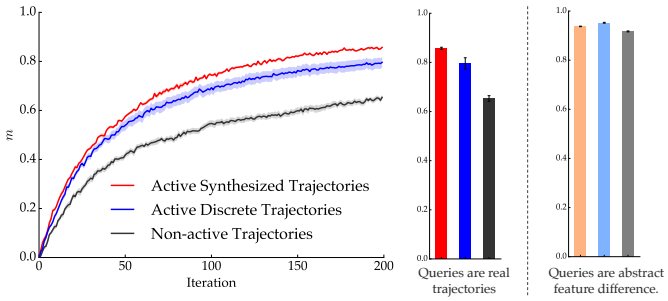


Fig. 4: A comparison of our algorithm (red) with a non-active (black) and a non-synthesis (blue) version. Both deciding on queries actively and synthesizing them instead of relying on a predefined set improve performance by a statistically significant margin. On the right, we run an experiment to argue that these things are really important when synthesizing real queries for continuous high-dimensional and constrained systems: the lighter colors are for a version that only generates a query in feature diference space, without rendering it into actual (feasible) trajectories – the algorithm is allowed to make a fake query. There, being active and especially doing synthesis don't matter (synthesis even hurts, likely because it is a local search and a discrete set of queries can better cover the space).

to perform synthesis is no longer useful compared to relying on a discrete set. Without needing to produce trajectories, a discrete set covers the much lower dimensional space of feature differences, and discrete search is not bottlenecked by local optima. This suggests that indeed, synthesis is important when we are dealing with learning reward functions for dynamical systems, requiring queries in a continuous, high-dimensional, and dynamically constrained space.

Overall, the results suggest that doing *active* query *synthesis* is important in reward learning applications of preference-based learning, with our results supporting both of our central hypotheses and demonstrating an improvement over prior methods which are either non-active or non-synthesis.

VII. Usability Study

Our experiments so far supported the value of our contribution: active preference-based learning that synthesizes queries from scratch outperformed both non-active and non-synthesis learning. Next, we ran a usability study with human subjects. The purpose of this study is not to compare our algorithm once again with other methods. Instead, we wanted to test whether a person can interact with our algorithm to recover a reasonable reward function for dynamical systems with continuous state and action spaces.

A. Experimental Design

In the learning phase of the experiments, we jumpstart the weights $\mathbf{w}$ with a reference starting distribution, and only update that based on the data collected for each user. We ask for 10 query inputs to personalize the distribution of $\mathbf{w}$ for each user.

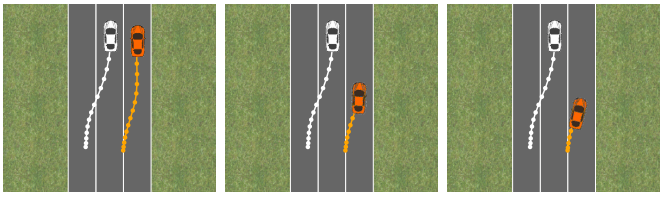


Fig. 5: Car trajectories (orange) in a scenario obtained by optimizing different weights. On the left, we see the trajectories optimized based on the learned weight $\mathbf{w}^*$, and the middle and right plots correspond to optimizing with $\mathbf{w}^1$ (slightly perturbed $\mathbf{w}^*$), and $\mathbf{w}^2$ (highly perturbed $\mathbf{w}^*$). The perturbations result in safe lane changes as well, but slower trajectories in this particular scenario.

Unlike our simulation experiments, here we do not have access to the ground truth reward function. Still, we need to evaluate the learned reward function. We thus evaluate what the users think subjectively about the behavior produced on a 7 point Likert scale.

Manipulated Factors. In order to calibrate the scale, we need to compare the reward's rating with some baseline. We choose to *perturb* the learned reward for this calibration. Thus we only manipulate one factor: *perturbation from the learned weights*. We ask users to compare trajectories obtained by optimizing for reward with three different values for the weights: i) $\mathbf{w}^* = \mathbb{E}[\mathbf{w}]$ providing the mode of the learned distribution from Alg. 1, ii) $\mathbf{w}^2$ a *large* perturbation of $\mathbf{w}^*$ which enables us to sanity check that the learned reward is better than a substantially different one, and iii) $\mathbf{w}^1$ a *slight* perturbation of $\mathbf{w}^*$ which is a harder baseline to beat and enables us to test that the learned reward is a local optimum. The weights thus vary in distance or alignment with the learned weight, from identical to very different. We simply add a zero mean Gaussian noise to perturb $\mathbf{w}^*$. For $\mathbf{w}^1$, we add a Gaussian with standard deviation of $0.1 \times |\mathbf{w}^*|$, and similarly with a standard deviation of $|\mathbf{w}^*|$ for $\mathbf{w}^2$. Our choice of perturbation is also affected by the test scenarios. In practice, we sample from possible $\mathbf{w}^1$ and $\mathbf{w}^2$ until the distance between the human trajectories is significant enough to create interesting scenarios that differentiate the weights (some simple scenarios have the car drive almost the same regardless of the reward function).

The 3 weights lead to 3 conditions. Fig. 5 shows one of the test environments where the robot (white car) decides to change lanes, and the human driver can take any of the three depicted orange trajectories. Here the trajectories correspond to the optimal human actions based on $\mathbf{w}^*$, $\mathbf{w}^1$, and $\mathbf{w}^2$ from left to right. The learned reward function results in a longer and faster trajectory where the human driver changes lane. The perturbations also result in a safe lane change; however, they result in much slower trajectories.

Dependent Measures. For 10 predefined scenarios (ini-

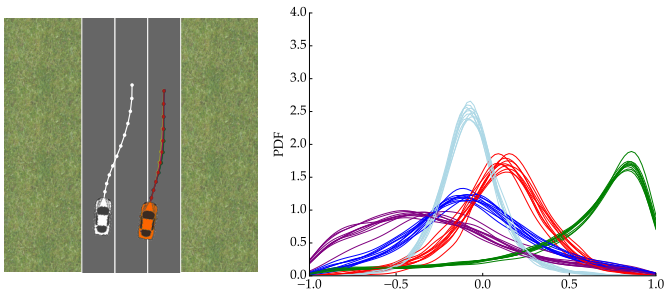


Fig. 6: Here we show the human trajectories of all users for a specific scenario based on optimizing the learned reward function $\mathbf{w}^*$. We plot the value of weights $\mathbf{w}^*$ for the five features for all users. This figure uses the same legend as Fig. 3.

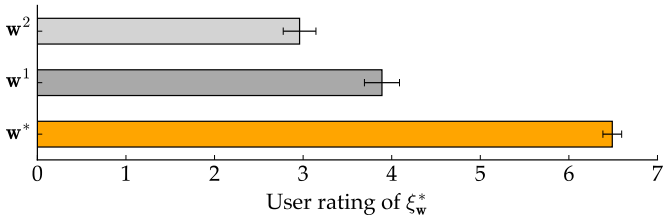


Fig. 7: User ratings. The orange bar corresponds to the learned weights $\mathbf{w}^*$, and the gray bars correspond to the perturbations of $\mathbf{w}^*$. Our users preferred $\mathbf{w}^*$ over the slightly perturbed ones $\mathbf{w}^1$, and preferred $\mathbf{w}^1$ over the highly perturbed ones $\mathbf{w}^2$.

tial state and actions of the other car $\mathcal{R}$), we generate the 3 trajectories corresponding to each condition. We ask users to rate each trajectory on a 7 point Likert scale in terms of how closely it matches their preference.

Hypothesis.

H3. *Perturbation from the learned weights negatively impacts user rating: the learned weights outperform the perturbed weights, with the larger perturbation result in the smallest rating.*

Subject Allocation. We recruited 10 participants (6 female, 4 male) in the age range of $26-65$. All owned drivers license with at least 8 years of driving experience. We ran our experiments using a 2D driving simulator, where we asked preference queries similar to Fig. 6.

B. Analysis

We ran an ANOVA with perturbation amount (measured via our metric m relative to the learned weights) as a factor and user rating as a dependent measure. Supporting H3, we found a significant negative effect of perturbation (positive effect of similarity m) on the user rating ($F = 278.2$, $p < .0001$). This suggests the learned weights are useful in capturing desired driving behavior, and going away from them decreases performance.

In Fig. 7, we show the aggregate result of the $1-7$ rating across all scenarios and users. As shown in this figure, the highest rating in orange corresponds to the learned reward weights $\mathbf{w}^*$, and our users preferred the slightly perturbed trajectories over the highly perturbed trajectories since the rating for $\mathbf{w}^1$ is higher than $\mathbf{w}^2$.

We found that, perhaps due to the fairly simple features we used, users tended to converge to similar weights, likely representing good driving in general. In Fig. 6, we show the learned weight distribution for every feature is shown for all 10 users, and the resulting optimal trajectories for all our users. These have very high overlap, further suggesting that users converged to similar reward functions. In future work, we aim to look at a more expressive feature set that enables people to customize their car to their individual desired driving style, as opposed to getting the car to just do the one reasonable behavior possible (which is what might have happened in this case). Nonetheless, the result is encouraging because it shows that people were able to get to a behavior that they and other users liked.

VIII. Discussion

Summary. We introduce an algorithm that can efficiently learn reward functions representing human preferences. While prior work relies on a discrete predefined set of queries or on passively learning the reward, our algorithm actively synthesizes preference queries: it optimizes in the continuous space of scenario parameters and trajectory pairs to identify a comparison to present to the human that will maximize the amount of expected volume that the answer will remove in the space of possible reward functions. We leverage continuous optimization to synthesize feasible queries, and assign a log-concave distribution over reward parameters to formulate an optimization-friendly criterion. We provide convergence guarantees for our algorithm, and compare it to a non-active and non-synthesis based approaches. Our results show that both active learning and synthesis are important. A user study suggests that the algorithm helps real end-users attain useful reward functions.

Limitations and Future Work. Our work is limited in many ways: we use a local optimization method for query synthesis which converges to local optima, we use an approximately rational model of the human when in fact real people might act differently, we assume that the robot has access to the right features, and we only consider one other agent in the world rather than multiple. Furthermore, our study shows that people can arrive at useful reward functions, such alignment with the reward leads to higher preference for the emerging trajectory, but it does not yet show that people can use this to customize their behavior – most users end up with very similar learned weights, and we believe this is because of the limitations of our features and our simulator. We plan to address these in the future.

Acknowledgments. This work was supported in part by the VeHICaL project (NSF grant #1545126).

References

[1] Nir Ailon and Mehryar Mohri. Preference-based learning to rank. *Machine Learning*, 80(2-3):189–211, 2010.

[2] Baris Akgun, Maya Cakmak, Karl Jiang, and Andrea L Thomaz. Keyframe-based learning from demonstration. *International Journal of Social Robotics*, 4(4):343–355, 2012.

[3] Riad Akrour, Marc Schoenauer, and Michèle Sebag. April: Active preference learning-based reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 116–131. Springer, 2012.

[4] Galen Andrew and Jianfeng Gao. Scalable training of l 1-regularized log-linear models. In *Proceedings of the 24th international conference on Machine learning*, pages 33–40. ACM, 2007.

[5] Chandrayee Basu, Qian Yang, David Hungerman, Anca Dragan, and Mukesh Singhal. Do you want your autonomous car to drive like you? In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, 2017.

[6] Darius Braziunas. Computational approaches to preference elicitation. *Department of Computer Science, University of Toronto, Tech. Rep*, 2006.

[7] Klaus Brinker, Johannes Fürnkranz, and Eyke Hüllermeier. Label ranking by learning pairwise preferences. Technical report, Technical Report TUD-KE-2007-01, Knowledge Engineering Group, TU Darmstadt, 2007.

[8] Christian Daniel, Malte Viering, Jan Metz, Oliver Kroemer, and Jan Peters. Active reward learning. In *Robotics: Science and Systems*, 2014.

[9] Johannes Fürnkranz, Eyke Hüllermeier, Weiwei Cheng, and Sang-Hyeun Park. Preference-based reinforcement learning: a formal framework and a policy iteration algorithm. *Machine learning*, 89(1-2): 123–156, 2012.

[10] Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486, 2011.

[11] Heikki Haario, Eero Saksman, and Johanna Tamminen. An adaptive metropolis algorithm. *Bernoulli*, pages 223–242, 2001.

[12] Rachel Holladay, Shervin Javdani, Anca Dragan, and Siddhartha Srinivasa. Active comparison based learning incorporating user uncertainty and noise.

[13] Ashesh Jain, Shikhar Sharma, Thorsten Joachims, and Ashutosh Saxena. Learning preferences for manipulation tasks from online coactive feedback. *The International Journal of Robotics Research*, 2015.

[14] Amin Karbasi, Stratis Ioannidis, et al. Comparison-based learning with rank nets. *arXiv preprint arXiv:1206.4674*, 2012.

[15] Sergey Levine and Vladlen Koltun. Continuous inverse optimal control with locally optimal examples. *arXiv preprint arXiv:1206.4617*, 2012.

[16] Manuel Lopes, Francisco Melo, and Luis Montesano. Active learning for reward estimation in inverse reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 31–46. Springer, 2009.

[17] László Lovász and Santosh Vempala. Fast algorithms for logconcave functions: Sampling, rounding, integration and optimization. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 57–68. IEEE, 2006.

[18] Constantin Rothkopf and Christos Dimitrakakis. Preference elicitation and inverse reinforcement learning. *Machine Learning and Knowledge Discovery in Databases*, pages 34–48, 2011.

[19] Dorsa Sadigh, Shankar Sastry, Sanjit Seshia, and Anca D. Dragan. Planning for autonomous cars that leverages effects on human actions. In *Proceedings of the Robotics: Science and Systems Conference (RSS)*, June 2016.

[20] Hiroaki Sugiyama, Toyomi Meguro, and Yasuhiro Minami. Preference-learning based inverse reinforcement learning for dialog control. In *INTERSPEECH*, pages 222–225, 2012.

[21] Aaron Wilson, Alan Fern, and Prasad Tadepalli. A bayesian approach for policy learning from trajectory preference queries. In *Advances in neural information processing systems*, pages 1133–1141, 2012.

[22] Christian Wirth, Johannes Fürnkranz, and Gerhard Neumann. Model-free preference-based reinforcement learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[23] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, pages 1433–1438, 2008.