Positive-Unlabeled Constraint Learning for Inferring Nonlinear Continuous Constraints Functions From Expert Demonstrations

Baiyu Peng and Aude Billard, *Fellow, IEEE*

***Abstract*—Planning for diverse real-world robotic tasks necessitates to know and write all constraints. However, instances exist where these constraints are either unknown or challenging to specify accurately. A possible solution is to infer the unknown constraints from expert demonstration. This letter presents a novel two-step Positive-Unlabeled Constraint Learning (PUCL) algorithm to infer a continuous constraint function from demonstrations, without requiring prior knowledge of the true constraint parameterization or environmental model as existing works. We treat all data in demonstrations as positive (feasible) data, and learn a control policy to generate potentially infeasible trajectories, which serve as unlabeled data. The proposed two-step learning framework first identifies reliable infeasible data using a distance metric, and secondly learns a binary feasibility classifier (i.e., constraint function) from the feasible demonstrations and reliable infeasible data. The proposed method is flexible to learn complex-shaped constraint boundary and will not mistakenly classify demonstrations as infeasible as previous methods. The effectiveness of the proposed method is verified in four constrained environments, using a networked policy or a dynamical system policy. It successfully infers the continuous nonlinear constraints and outperforms other baseline methods in terms of constraint accuracy and policy safety.**

***Index Terms*—Learning from demonstration, reinforcement learning, transfer learning.**

I. Introduction

PLANNING for many robotics and automation tasks requires explicit knowledge of constraints, which define which states or trajectories are allowed or must be avoided [1], [2]. Sometimes these constraints are initially unknown or hard to specify mathematically, especially when they are nonlinear, possess unknown shape of boundary or are inherent to user's preference and experience. For example, human users may determine an implicit minimum distance of the robot to obstacles based on the material of the obstacle (glass or metal), the shape and density of obstacles and their own personal preference. The robot should be able to infer such a constraint to achieve the task goal and meet human's requirements. This means an explicit constraint should be inferred somehow, e.g., from existing human demonstration sets.

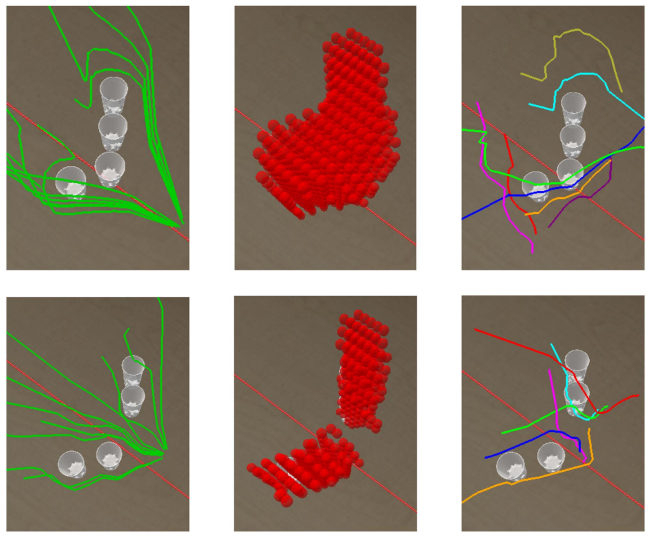


Fig. 1. Learning an obstacle-avoidance constraint from demonstrations with proposed method. The task requires the robot to reach a target state while avoiding going close to or over the cups. These unknown constraints can be translated into an infeasible region that must be inferred from demonstrations. The top row illustrates one obstacle configuration, while the bottom row shows another. In each row, the left image displays the expert demonstrations (in green), the middle image shows the learned constraint region (in red), and the right image illustrates the learned policy, which is trained from the learned constraint and tested on a set of shifted goal states and starting points.

Constraint inference from demonstrations has drawn more and more attention since 2018 [3]. Various methods have been developed to learn various types of constraints. In this work, we focus on the constraints of not visiting some undesired states(-actions) throughout the trajectory (discussed in detail in Section II-A). These constraints define what the user does not want to happen and are ubiquitous in practice, e.g., not crashing into an unknown obstacle or not surpassing an unknown maximum velocity.

Existing constraint learning methods can be roughly divided into inverse reinforcement learning (IRL)-based methods and mixed-integer-programming (MIP)-based methods.

IRL-based methods draw inspiration from the concepts and approaches of IRL and adapt them to learn the constraint function. [4] first formulate the constraint learning problem in the

Received 24 July 2024; accepted 12 December 2024. Date of publication 25 December 2024; date of current version 7 January 2025. This article was recommended for publication by Associate Editor S. Bansal and Editor A. Faust upon evaluation of the reviewers' comments. This work was supported in part by euROBIN and in part by ERC SAHR Grant. *(Corresponding author: Baiyu Peng.)*

The authors are with the LASA, School of Engineering, EPFL (Swiss Federal Institute of Technology in Lausanne), 1015 Lausanne, Switzerland (e-mail: baiyu.peng@epfl.ch; aude.billard@epfl.ch).

All codes are available at https://github.com/epfl-lasa/PUCL_learning_constraint.

Digital Object Identifier 10.1109/LRA.2024.3522756

2377-3766 © 2024 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: EPFL LAUSANNE. Downloaded on October 26,2026 at 09:32:26 UTC from IEEE Xplore. Restrictions apply.

maximum likelihood inference framework. They introduce a Boltzmann policy model, where the likelihood of any feasible trajectory is assumed to be proportional to the exponential return of the trajectory, while the likelihood of any infeasible trajectory is 0. Then a greedy algorithm is proposed to add the smallest number of infeasible states that maximize the likelihood of the demonstrations. This framework has the advantage of being able to work with sub-optimal demonstrations. Further, [5] and [6] extend the maximum likelihood framework from a deterministic setting into a stochastic setting, with soft or probabilistic constraints. [7] proposes to learn a neural network policy via deep RL and identify discrete infeasible states with the policy network. All aforementioned methods only extract constraint from a pre-defined finite constraint set. To learn an arbitrary continuous constraint function, [8] proposes Maximum Entropy Constraint Learning (MECL) algorithm. It not only approximates the policy with a network but also learns a constraint network, which is optimized by making a gradient ascent on the maximum likelihood objective function. More recently, [9] proposes a variational approach to infer the distribution of constraint and capture the epistemic uncertainty. However, despite that [8], [9] can learn a continuous constraint function, they have following shortcomings: theoretically, their loss function optimizes over all the states visited by the policy to be infeasible, and states visited by the demonstrations to be feasible. If demonstrations and policy overlap, this loss will lead to conflicts during the update step, slowing the training time and prone to classifying safe demonstrated states as infeasible. In addition to that, in practice [8], [9], [10] are used to recover linear constraints, and on a unidimensional state, e.g. learn a constraint of the type $x \geq -3$, and, as we show later in the letter, perform relatively poorly when learning nonlinear constraints.

In parallel, another line of research based on MIP has been developed. [3] first studies how to infer infeasible states from a demonstration set in a grid world. They assume that all possible trajectories that could earn higher rewards than the demonstration must be constrained in some way, or the expert could have passed those trajectories. In practice, they use hit-and-run sampling to obtain such higher-reward trajectories and solve an integer program to recover the constrained states in a grid world. This MIP method is soon extended to continuous state space and parameterized continuous constraint functions, where the constraint is represented as unions and intersections of axis-aligned boxes [11]. Later, the same authors propose to leverage the closed-form environmental model and the KKT optimality condition to infer the parameters of a known constraint parameterization [11], or a Gaussian Process constraint model [12]. However, these MIP-based methods either require the full knowledge of a differentiable model of the environment and agent [11], [12], which is not always available in practice, or can only learn constrained region expressed with unions and intersections of box [13].

In summary, despite these impressive advances, it still remains to be shown how to learn continuous nonlinear constraint functions, without full knowledge of true constraint parameterization and environmental model. To overcome the aforementioned challenge, we propose a two-step Positive-Unlabeled Constraint Learning (PUCL) method, inspired by a machine learning subarea Positive-Unlabeled (PU) learning [14]. Within our framework, we treat all data in demonstrations as positive (feasible) dataset, and learn a policy to generate many potentially unsafe trajectories which serve as unlabeled data with unknown feasibility. Then a two-step PU learning technique is applied, where we first identify reliable infeasible data and secondly learn a binary feasibility classifier (i.e., constraint function) from the positive data and reliable infeasible data. The proposed method is flexible to learn complex-shaped constraint boundary without the need to know the true constraint parameterization. It also avoids the drawback of some IRL-based methods that mistakenly optimize and classify demonstrations as infeasible, thus improving the performance and accuracy.

II. Preliminaries and Problem Statements

A. Preliminaries and Notations

We consider a finite-horizon deterministic Markov decision process (MDP) [15], where the states and actions are denoted by $s \in \mathcal{S}$ and $a \in \mathcal{A}$. γ denotes the discount factor, and the real-valued reward is denoted by $r(s, a)$. A trajectory $\tau = \{s_1, a_1, \ldots, s_T, a_T\}$ contains a sequence of state-action pairs in one episode. For the trajectory τ, the total discounted reward (return) is defined as $r(\tau) = \sum_{t=1}^{T} \gamma^t r(s_t, a_t)$. A policy, the mapping between states and actions, is denoted by $a = \pi(s)$.

In this study, we focus on learning the Markovian state-action constraints, i.e., avoiding visiting some forbidden states (and/or actions) throughout the entire trajectory. Mathematically, we denote the true constraint set by $\mathcal{C}^* = \{(s, a) \in \mathcal{S} \times \mathcal{A} | (s, a) \text{ is truly infeasible}\}$. Thus, the true constraint is written as $(s, a) \notin \mathcal{C}^*$. This definition implicitly assumes that the studied constraint is (1) time-independent; (2) deterministic: a state-action pair is either truly feasible or infeasible. In addition to the unknown constraints, some known constraints could also be present in the task, e.g., goal state constraints and system dynamic constraints. For simplicity of explanation, all demonstrations and trajectories are assumed to satisfy the known constraints (trajectories violating known constraints are discarded and not used for inferring the unknown constraint).

We will approximate the true constraint set with a constraint network ζ_θ, with θ as the parameter. To simplify the notation and allow for more flexibility, we first introduce a generalized state $\bar{s} \equiv \psi(s, a)$ as the input to the network, where $\psi(s, a)$ is a user-defined feature function that can be customized to incorporate prior knowledge about the constraint structure. For example, if the constraint is known to depend solely on the state (or actions), one could define $\psi(s, a) = s$ (or a). Readers should keep in mind that, in the following sections, we use the generalized state $\bar{s}$ as shorthand for $\psi(s, a)$. Following this notation, the constraint network is written as $\zeta_\theta(\bar{s}) \in [0, 1]$, which induces a constraint set $\mathcal{C}_\theta = \{(s, a) | \zeta_\theta(\bar{s}) \leq d\}$ with $d = 0.5$ being the decision threshold.

The demonstration set $\mathcal{D}_\tau = \{\tau_i\}_{i=1}^{M_d}$ comprises trajectories demonstrated by a demonstrator. For convenience, we also introduce the set $\mathcal{D} = \{\bar{s} | (s, a) \in \tau_i \in \mathcal{D}_\tau\}$ consisting of generalized states from all the demonstrated trajectories. Similarly, we define

Authorized licensed use limited to: EPFL LAUSANNE. Downloaded on October 26,2026 at 09:32:26 UTC from IEEE Xplore. Restrictions apply.

the unlabeled data set $\mathcal{P}_\tau = \{\tau_i\}_{i=1}^{M_p}$ and $\mathcal{P} = \{\bar{s}|(s,a) \in \tau_i \in \mathcal{P}_\tau\}$.

B. Problem Formulation

As in similar works [3], [8], we require that the reward function to be known in advance. This assumption is reasonable in many robotics applications, where the reward might be shortest path, shortest time to reach the goal. We emphasize that the unknown constraints are not indicated in the known reward function, i.e., there is no negative reward for violating the unknown constraints. The real task is written as a constrained optimization problem:

Problem 1 (Constrained optimization problem)

$$\max_{\{a_i \sim \pi\}_{i=0}^{T}} \quad r(\tau) \tag{1a}$$

$$\text{s.t.} \quad (s_i, a_i) \notin \mathcal{C}^*, \quad \forall\, (s_i, a_i) \in \tau \tag{1b}$$

where $r(\tau)$ represents the reward function of trajectory τ. We make the following assumption of the demonstrations optimality and feasibility:

Assumption 1 (Demonstrations Sub-optimality and Feasibility): Any demonstration $\tau_i \in \mathcal{D}_\tau$ is feasible and δ-sub-optimal solution to the problem 1:

1) (feasible): τ_i satisfies (1b) and all known constraints.
2) (δ-sub-optimal): $(1-\delta)r(\tau^*) \leq r(\tau_i)$, where τ^* is the feasible optimal solution to the problem 1. $\delta \in [0, 1)$ is a coefficient of sub-optimality.

In practice δ is treated as a hyper-parameter and specified according to the user's confidence of demonstration optimality. We formally define the constraint learning task as:

Definition 1 (Constraint Learning): The task of learning constraint from demonstration is to recover the unknown true constraint set C^* in Problem 1 from the demonstration set $\mathcal{D}_\tau$, given the known reward function $r(\tau)$.

III. Method

A. Motivation: Constraint Inference as Positive-Unlabeled Learning

As in many IRL-based and MIP-based works [3], [8], we infer the underlying constraint by contrasting the (safe) demonstration with some potentially unsafe trajectories generated by a certain policy. The training follows an iterative framework to incrementally learn the constraint. In each iteration, we first sample a set of high-reward trajectories $\mathcal{P}_\tau$ by performing the current policy π_ϕ (discussed later in III-D). Similar to [3], we speculate that the trajectory which wins a higher reward than demonstration is potentially unsafe and cheats by visiting some forbidden states. However, it remains unclear which specific state(s) within the trajectory are unsafe. In other words, trajectory visits both truly feasible states and truly infeasible states but all states remain unlabeled. In contrast, it is certain that all states on the demonstrated trajectories are labeled feasible by Assumption 1. Our goal is to classify each single state as feasible or infeasible by learning from a batch of fully labeled feasible states and another batch of unlabeled mixed states.

This insight inspires us to formulate constraint inference as a positive-unlabeled learning problem. PU learning is a machine learning technique that aims to learn a classifier from a positive dataset and an unlabeled dataset containing both positive and negative samples [14]. Within our framework, positive samples correspond to feasible states. Thus, the data from demonstrations serve as positive data, while the policy offers unlabeled data containing some infeasible samples. We can thus utilize PU learning methods to uncover the infeasible states from the two datasets.

PU learning methods for binary classification mainly divide into three categories: two-step methods, biased learning and weight-based methods [14]. The latter two methods both rely on an important assumption, Sampled Completely At Random (SCAR), which requires the demonstrations and policy to have a similar distribution in the truly feasible region. This SCAR assumption could be unrealistic and limits its application to only specific environments. In contrast, the two-step methods do not require SCAR assumption and only require the two classes can be strictly separated. Since we limit our discussion to deterministic constraint, i.e., a state is either truly feasible or infeasible, this separability assumption naturally holds. Therefore, this letter will develop a constraint learning method from a two-step framework [16], [17].

B. Method

Our two-step PU learning method consists of the following steps: (1) identifying reliable infeasible data (denoted as $\mathcal{R}$) from unlabeled set $\mathcal{P}$, and (2) using supervised learning techniques to learn a classifier from the feasible demonstrations $\mathcal{D}$ and reliable infeasible dataset $\mathcal{R}$.

In step 1, the identification of $\mathcal{R}$ starts from the intuition that data in the unlabeled set $\mathcal{P}$ that are very different from all data in the demonstration set $\mathcal{D}$ are highly likely to be truly infeasible. We propose a kNN-like distance-based approach, which is straightforward to comprehend and implement, to identify the reliable infeasible set $\mathcal{R}$. The unlabeled data are first ranked according to their distance to the k nearest feasible data from demonstration set.[1] Mathematically, the ranking score for each unlabeled data $\bar{s} \in \mathcal{P}$ is calculated by the average distance:

$$S_{\mathcal{D}}(\bar{s}) = \frac{1}{k} \sum_{\bar{s}' \in kNN(\bar{s}, \mathcal{D})} dis\,(\bar{s}, \bar{s}') \tag{2}$$

where $kNN(\bar{s}, \mathcal{D})$ indicates the k-nearest neighbors of $\bar{s}$ in the set $\mathcal{D}$, and $dis(\bar{s}, \bar{s}')$ is a distance metric selected based on the specific task. Then, the unlabeled data whose scores are higher than a threshold d_r are selected as reliable infeasible data:

$$\mathcal{R} = \{\bar{s} \in \mathcal{P} | S_{\mathcal{D}}(\bar{s}) \geq d_r\} \tag{3}$$

Intuitively d_r indicates user's belief over the size of true infeasible region. A lower d_r tends to identify a larger $\mathcal{R}$, leading to a more conservative constraint network. Conversely, a higher d_r tends to find a smaller $\mathcal{R}$ and a more radical constraint function.

[1] Since $\mathcal{D}$ contains consecutive data points from trajectories, it is recommended to use a small k around 1-3.

Authorized licensed use limited to: EPFL LAUSANNE. Downloaded on October 26,2026 at 09:32:26 UTC from IEEE Xplore. Restrictions apply.

In the extreme case of $d_r = 0$, nearly any state not visited by the demonstrations is classified as infeasible. We admit selecting d_r can be tricky, but such a parameter is generally inevitable since learning constraints is known to be an ill-posed problem with infinite valid solutions [8].

In the case where all dimensions of $\bar{s}$ are homogeneous, i.e., having similar physical meanings and units, d_r will share these units and can be selected heuristically. For example, the first case in our experimental section learns an obstacle-avoidance constraint in a 3D position space and d_r roughly indicates how far the demonstration is from the true obstacle.

While in some high-dimensional case where $\bar{s}$ contains heterogeneous dimensions, identifying $\mathcal{R}$ with distance-based metric and determining the corresponding d_r are more tricky. We propose to merge $\mathcal{D}$ and $\mathcal{P}$, standardize each dimension of the combined dataset to have zero mean and unit variance, and then use the standardized data to compute the scores $S_{\mathcal{D}}(\bar{s})$. Based on these scores, we select a percentile X and identify the top X percent of the highest-scoring data in $\mathcal{P}$ as the reliable infeasible data.

In practice, we also notice that when the demonstration closely adheres to the true constraint boundary at a short distance, kNN-like metric may fail to learn an accurate boundary, as the reliable infeasible data are all at least d_r far from the demonstrations. To adapt to this case, we expand $\mathcal{R}$ by adding the state from each trajectory in $\mathcal{P}$ that is closest to the original $\mathcal{R}$, since these data are likely to lie between the current boundary and the true boundary.

$$\mathcal{R} \leftarrow \mathcal{R} \cup \left(\cup_{\tau_i \in \mathcal{P}_\tau} \arg \min_{(s,a) \in \tau_i} S_{\mathcal{R}}(\bar{s}) \right) \tag{4}$$

where $S_{\mathcal{R}}(\bar{s})$ is the score function sharing similar definition as $S_{\mathcal{D}}(\bar{s})$ in (2) but with a different set $\mathcal{R}$. Note that this expansion is performed only once per iteration.

In step 2, we train a neural network binary classifier $\zeta_\theta(\bar{s})$ from the reliable infeasible data $\mathcal{R}$, the memory buffer $\mathcal{M}$ (discussed later in Section III-C) and the (feasible) demonstrations data $\mathcal{D}$ with a standard cross entropy loss:

$$\mathcal{L}(\theta) = -\frac{1}{N+M} \left[\sum_{\bar{s}_i \sim \mathcal{D}}^{N} \log \zeta_\theta(\bar{s}_i) - \sum_{\bar{s}_j \sim \mathcal{R} \cup \mathcal{M}}^{M} \log\left(1 - \zeta_\theta\left(\bar{s}_j\right)\right) \right] \tag{5}$$

C. Iterative Learning Framework With Policy Filter and Memory Buffer

Last subsection discussed learning constraint from a given unlabeled set $\mathcal{P}$ and demonstrations $\mathcal{D}$. This subsection and the next will elaborate on the generation of unlabeled set $\mathcal{P}$. Existing methods generate $\mathcal{P}$ either in an iterative manner [4], [8], [9] or in a one-batch manner [3], [7], [13]. The former iteratively updates policy with current constraint function and generates new trajectories to update the constraint. Since the policy is always updated with respect to the up-to-date constraint network, the new trajectories generated in each iteration are mostly distributed around the boundary of the current constraint network, which will improve the accuracy of the constraint boundary. Therefore, this manner is efficient to learn the constraint boundary and can automatically adjust the distribution of the generated trajectories. In contrast, the one-batch manner generates a bulk of trajectories uniformly [3] or greedily [7] at the start of training. The following training takes place only on those trajectories, with no new data generated. This manner is more stable but relatively inefficient and poses a higher demand on the diversity and the coverage of the generated trajectories.

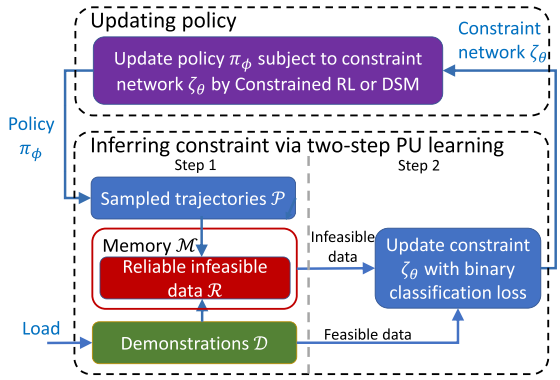


Fig. 2. The framework of the proposed constraint learning algorithm PUCL. It alternates between updating policy using current constraint network, and inferring constraint from demonstrations and current policy. The constraint is inferred using a novel two-step positive-unlabeled learning technique. In the first step, the trajectories $\mathcal{P}$ sampled from current policy are viewed as unlabeled data, while the demonstrations are viewed as labeled feasible data. From the two datasets we identify reliable infeasible data using a distance-based metric. In the second step, the reliable infeasible data from current iteration and previous iterations, as well as the feasible demonstrations, are used to train a constraint network using a standard binary classification loss.

This work follows the iterative framework, and introduces a policy filter and a memory buffer. Fig. 2 gives a sketch of the whole iterative structure.

Policy filter: In each iteration of policy updating, the policy is updated for fixed steps with the current constraint network $\zeta_\theta(\bar{s})$. To prevent undesired updates from poor policies, a policy filter (6) is introduced to select trajectories with a relatively high reward than the demonstration, which are believed to truly violate the unknown constraint according to Assumption 1.

$$\mathcal{P}_\tau \leftarrow \left\{ \tau_i \in \mathcal{P}_\tau | (1-\delta) r(\tau_i) \geq r\left(\tau_i^{\mathcal{D}}\right) \right\} \tag{6}$$

where δ is a coefficient defined in Assumption 1, $\tau_i^{\mathcal{D}}$ is the demonstration that starts from the same state as τ_i.

Memory buffer: Most iterative-framework-based papers generate new trajectories in each iteration and discard those from previous iterations [4], [8], [9]. Such a training manner tends to forget the infeasible region learned in early iterations. Thus, we introduce a memory replay buffer and record all the reliable infeasible data $\mathcal{R}$ identified in each iteration into the memory buffer $\mathcal{M}$. In the following iterations of learning constraints, both the current $\mathcal{R}$ and memory buffer $\mathcal{M}$ serve as infeasible data for training (see (5)).

D. Represent and Learn Policy Via Constrained RL or Dynamical System Modulation

In the iterative framework discussed in the last subsection, we need to maintain a policy to generate high-reward trajectories while satisfying the currently learned constraints. We consider and compare two policies: 1) constrained RL [18], which is

Authorized licensed use limited to: EPFL LAUSANNE. Downloaded on October 26,2026 at 09:32:26 UTC from IEEE Xplore. Restrictions apply.

effective at learning constrained policies in a high-dimensional environment with complex reward function but suffers from a slow and unstable training process, and 2) dynamical system modulation (DSM) [19], which modulates the nominal policy by multiplying a rotation matrix to avoid infeasible regions. Unlike RL, the modulation requires no explicit training of the policy. Thus, DSM minimizes training time and provides a formal guarantee that the policy will never violate the learned constraints. However, DSM is applicable only for tasks where the control command is the velocity of each state, e.g., controlling the end-effector pose of a robot arm.

Constrained reinforcement learning: A networked policy is trained with a modern constrained RL algorithm PID-Lagrangian [18], which offers the advantage of automatically and more stably adjusting the penalty weight. As shown in (7), it reshapes the original reward by incorporating the constraint as a penalty term into the original reward function to avoid the infeasible states. Here, w_p is a penalty weight adjusted by PID-Lagrangian updating rules [18], [20], [21], and $c(s,a)$ is the constraint indicator function. Using this reshaped $r'(s,a)$, a constrained optimal policy can be straightforwardly learned with the standard RL algorithm PPO [22].

$$r'(s,a) = r(s,a) - w_p c_\theta(s,a), \quad \text{where}$$
$$c_\theta(s,a) = \begin{cases} 0 \text{ if } \zeta_\theta(\bar{s}) > d \ \text{ (feasible)} \\ 1 \text{ if } \zeta_\theta(\bar{s}) \leq d \ \text{ (infeasible)} \end{cases} \tag{7}$$

Dynamical system modulation: DSM first learns a nominal unconstrained policy and later modulates nominal policy with the constraint network. The nominal dynamical system policy is represented as (8), where s_g is the goal state and matrix $A(s)$ is a Gaussian Mixture Regression model learned from the known reward function and preserves stability, following [23].[2]

$$\pi_n(s) \equiv \dot{s} = A(s)(s - s_g) \tag{8}$$

Given a learned constraint network $\zeta_\theta(s)$ with state-only constraint, (9) produces a provably safe policy[3] $\pi(s)$ by multiplying a rotation matrix $M(s)$ constructed from $\zeta_\theta(s)$ following [19].

$$\begin{aligned} \pi(s) &= M(s)\pi_n(s), \quad \text{where} \\ M(s) &= E(s)D(s)E^{-1}(s), \\ E(s) &= [r(s), e_1(s), e_2(s), \ldots e_{n-1}(s)], \\ n(s) &= \frac{d\zeta(s)}{ds}, n(s) \perp e_1(s) \perp e_2(s) \perp \ldots e_{n-1}(s) \\ D(s) &= diag\left[\lambda_1(s), \lambda_2(s), \lambda_3(s) \ldots, \lambda_n(s)\right], \\ \lambda_1(s) &= 1 - \frac{1}{\Gamma(s)}, \lambda_{2,3,\ldots,n}(s) = 1 + \frac{1}{\Gamma(s)}, \\ \Gamma(s) &= 1 + 10 * (\zeta(s) - 0.5)^{0.2} \end{aligned} \tag{9}$$

[2]In our experiment, for DSM baseline we directly set $A(s) = I$ since the task is reaching the destination in the shortest path, which is a straight line connecting the current state and the goal state.

[3]The DS modulation theory guarantees that the modulated policy will never violate the learned constraint. Additionally, if the true constraint is a star-shaped obstacle and the nominal policy is linear, the modulated policy also remains globally stable [19].

Algorithm 1: Positive-Unlabeled Constraint Learning (PUCL).

input: Demonstration set $\mathcal{D}_\tau$, randomly initialized constraint network ζ_θ, randomly initialized policy network π_ϕ, empty memory buffer $\mathcal{M} = \varnothing$
repeat
  % Learning policy
  Evaluate current policy π_ϕ and adapt the penalty weight w_p in (7) through PID-Lagrangian [18]
  Update the policy π_ϕ with the current constraint network ζ_θ using constrained RL or DSM
  % Learning constraint with two-step PU learning
  Sample trajectory set $\mathcal{P}_\tau$ using the current policy π_ϕ from the same starting points as each demonstration
  Filter sampled set $\mathcal{P}_\tau$ with (6):
  Identify and expand reliable infeasible set $\mathcal{R}$ using (3) and (4)
  Update the constraint ζ_θ with the demonstration $\mathcal{D}$, reliable infeasible set $\mathcal{R}$ and memory buffer $\mathcal{M}$ with (5)
  Save all data from $\mathcal{R}$ into the memory buffer $\mathcal{M}$
until reach maximum iterations

where $n(s)$ is the normal vector to the constraint boundary, $e_i(s)$ are the corresponding tangent vectors, $r(s)$ is the vector towards a reference point inside the obstacle, $\lambda_i(s)$ modifies the velocity along different directions, and $\Gamma(s)$ specifies the relative distance of current state s to the constraint boundary. Intuitively, for any state s close to the constraint boundary, $\Gamma(s)$ approaches 1 and $\lambda_1(s)$ approaches 0, resulting in zero normal velocity and preventing the agent from penetrating the obstacle.

The pseudocode of the proposed method is attached below.

IV. Experiment

A. Comparison of Constraint Learning Methods

To examine the performance of the proposed method, we apply it to four simulated environments to learn a 2D position constraint, a 3D position constraint, a 3D control input constraint and a constraint of an 18D cheetah robot, respectively. All codes are available at https://github.com/epfl-lasa/PUCL_learning_constraint.

Baselines: Four constraint learning methods are compared here, all using constrained RL as the policy: 1) PUCL (the proposed method); 2) maximum-entropy constraint learning (MECL) [8], a popular IRL-based method that can learn a continuous arbitrary constraint function; 3) binary classifier (BC) [8], a method directly trains an NN feasibility classifier using loss (5) but with all data from $\mathcal{P}$ as infeasible data; 4) generative-PUCL (GPUCL), an alternative PUCL approach using a generative model to identify reliable infeasible data. It first learns a Gaussian Mixture Model (GMM) of the expert distribution, and then identifies reliable infeasible data as the data from $\mathcal{P}$ with the lowest likelihood of being generated by the GMM. GPUCL is also first formulated in this letter but we do not discuss in detail due to its weaker performance.

Authorized licensed use limited to: EPFL LAUSANNE. Downloaded on October 26,2026 at 09:32:26 UTC from IEEE Xplore. Restrictions apply.

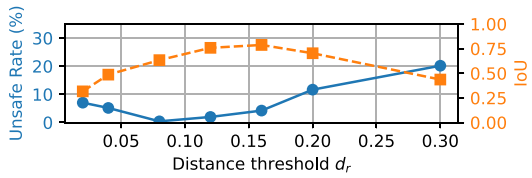


Fig. 3. Sensitivity analysis of the distance threshold d_r on performance. The left y-axis represents the unsafe rate (lower is better), and the right y-axis represents the IoU (higher is better). The curve is averaged on 5 runs with different random seeds.

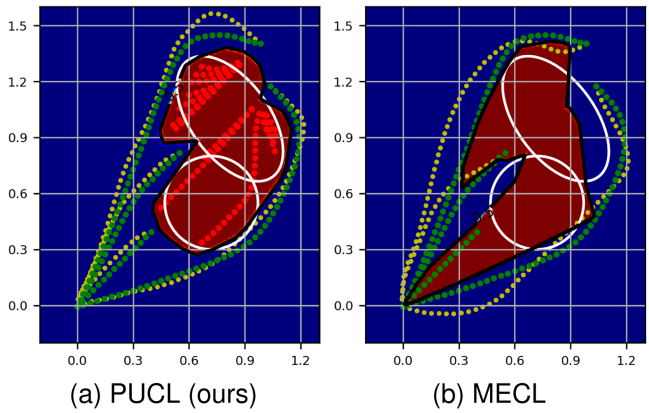


Fig. 4. True constraint and constraint learned with the proposed method PUCL and baseline MECL [8] in the 2D reaching task. The two white ellipses represent the boundary of the true constraint. The heat map is a visualization of the learned constraint ζ_θ, where the red region of $\zeta = 0$ represents infeasibility, while the blue regions of $\zeta = 1$ indicate feasibility. The trajectories of the demonstrations and the policy are shown in green and yellow, respectively. The red points correspond to identified reliable infeasible data $\mathcal{R} \cup \mathcal{M}$ (only for PUCL method).

Metrics: we present the evaluation utilizing two metrics: 1) the IoU (the intersection over union) index, which measures the correctness of the learned constraints. We uniformly sample points in the state space, and compute IoU as (Number of points which are both predicted to be infeasible and truly infeasible) divided by (Number of points either predicted to be infeasible or truly infeasible); the constraint models of all four methods use 0.5 as the decision threshold. 2) the unsafe rate, the per-step true constraint violation rate of the policy learned from the learned constraint.

2D/3D Position Constraints: We first study the two obstacle-avoidance constraints of the end-effector of a robot arm, one in 2D position space and another in 3D position space. We defined a goal-reaching task, where the agent is initialized randomly and rewarded to reach a goal state while staying away from the unknown obstacles. The state space consists of 2D/3D coordinates of the end-effector and the control variables are its corresponding 2D/3D velocities, capped at 0.58 m/s. The reward functions in both tasks are the negative of the Cartesian path distance to reach the goal, a natural choice for a goal-reaching task. The constraint networks in both tasks take only the state as input, i.e., $\bar{s} = s$.

In the 2D reaching task, the true constraint involves an irregular obstacle composed of two ellipses as shown in Fig. 4. In the 3D reaching task, the true constraint is avoiding the tall cylinders. We stress that learning such constraints in our setting is nontrivial and has not been well addressed by prior works. Specifically, our setting does not assume prior knowledge of the obstacles size, shape, or location, while relevant studies [13] typically require a known parameterization of obstacles and only infer a handful of parameters (e.g., height and radius of cylinders). Furthermore, the studied constraints are nonlinear, not just boxes or hyperplanes, which contrasts with previous works that only learn a plain constraint of a single dimension such as coordinate $x \geq -3$ [8], [9]. Finally, unlike [12], our method does not require a differentiable environmental model.

TABLE I
LIST OF HYPER-PARAMETERS OF PUCL

	2D task	3D task
Policy network	64, 64	64, 64
Value network	64, 64	64, 64
Constraint network	32, 32	32, 32
Policy learning rate	3×10^{-4}	3×10^{-4}
Constraint learning rate	5×10^{-3}	5×10^{-3}
Distance metric	Euclidean	Euclidean
$\mathcal{R}$ distance threshold d_r	0.12	0.03
$\mathcal{R}$ identification k	1	1
sub-optimal coefficient δ	0	0.03

The expert demonstration sets for 2D and 3D tasks consist of 4 and 10 safe trajectories, respectively. All demonstrations are generated by an entropy-regularized suboptimal RL agent, trained assuming full knowledge of the true constraint.

A list of important hyper-parameters of our algorithm is given in Table I. For the neural network architectures, the number of hidden units in each layer is mentioned. The hidden activation functions in all networks are Leaky ReLU. The output activation function of the constraint network is the sigmoid function, whereas that of the policy and value network is tanh. Other baselines all adopt identical network architecture.

The IoU index and the unsafe rate in the two tasks are presented in Fig. 5. Each group of 10 runs uses the same demonstrations and hyper-parameters, but differ in the initialization parameters for the networks. The proposed method PUCL exhibits superior performance compared to the baseline across both environments and metrics. It achieves a near-zero unsafe rate at the convergence of the training in both environments and outperforms baseline MECL. GPUCL adopts a similar framework to PUCL but performs worse than PUCL, possibly due to the difficulty of fitting a good expert distribution with a very limited numbers of demonstrations.

To provide readers with an intuitive understanding, we include visualizations of the 2D constraint learned with PUCL and MECL in Fig. 4. Comparing the learned constraints (red region) with the true constraints (white ellipses) confirms the effectiveness of our method in acquiring a model of a nonlinear constraint. The baseline method MECL learns a less accurate constraint, and even some states visited by the demonstrations are classified as infeasible.

While our method is effective, it can have a relatively strong dependency on the distance threshold d_r. To clearly illustrate the impact of this hyperparameter on performance, we present a sensitivity analysis by varying d_r in the 3D task (see Fig. 3).

Authorized licensed use limited to: EPFL LAUSANNE. Downloaded on October 26,2026 at 09:32:26 UTC from IEEE Xplore. Restrictions apply.

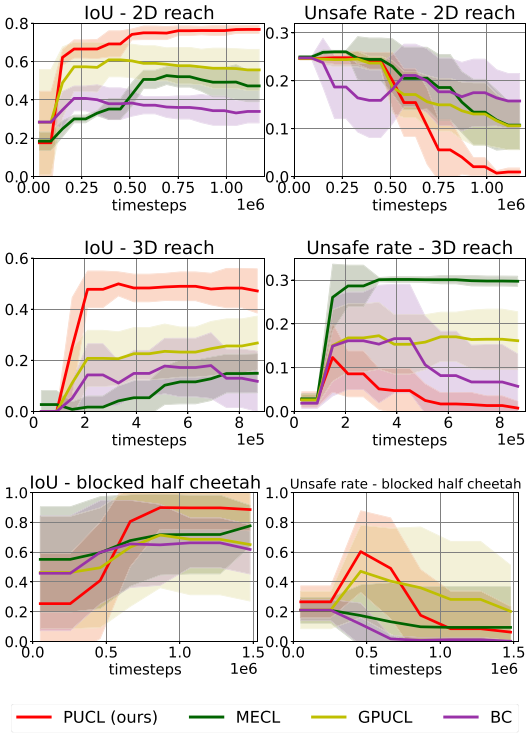


Fig. 5. The IoU index (higher is better) and unsafe rate (lower is better) in three environments (top: 2D reaching, middle: 3D reaching, bottom: blocked half cheetah). The x-axis indicates the training process, the number of timesteps the agent takes in the environment. All the results are the average of 10 independent runs, and the shaded area represents one standard deviation.

Control Input Constraint: To demonstrate our method's ability to learn control input constraint, we modify the 3D reaching task by imposing stricter maximum velocity constraints. The original permitted velocities limits are $|v_x|, |v_y|, |v_z| \leq 0.58$ m/s. Imagine that when the human users demonstrate the motion, they may expect the robot to move slower according to their personal preference, e.g., a stricter velocity constraint $|v_x| \leq 0.48$ m/s, $|v_y| \leq 0.58$ m/s, $|v_z| \leq 0.19$ m/s. The agent is tasked with inferring this user-specific velocity constraint from demonstrations. Importantly, although the true constraints follow a box form, the agent is not given any prior knowledge of this structure; it cannot assume a box constraint and simply find maximum values from the data. For simplicity, the obstacle-avoidance constraint is assumed to be known in this task, and only the velocity constraint is learned, thus $\bar{s} = a$.

We collected 30 demonstrations from an expert RL agent and applied PUCL and MECL to learn the velocity constraint. For PUCL, we used the Euclidean distance with a threshold of $d_r = 0.01$. The results, averaged over 8 runs, showed that PUCL achieved an IoU score of 0.87 ± 0.08, compared to 0.77 ± 0.26 for MECL. In terms of unsafe rate, PUCL achieves an unsafe rate of only $(3.47 \pm 3.26)\%$ compared to $(28.1 \pm 19.2)\%$ for MECL. Furthermore, the velocity distribution output by the policy learned with PUCL is shown in Fig. 6, demonstrating that almost all data points fall within the true safe range.

Constraint in high-dimensional space: The previous tasks extracts constraints only from 2D/3D spaces. To showcase our method extends to high-dimensional, heterogeneous state space, we apply it to an 18D blocked half cheetah environment introduced by baseline papers [8], [9]. The agent controls a cheetah robot with two legs by applying torques on each joint motors. The reward is determined by the distance it walks between the current and the previous time step. The constraint blocks the region with x-coordinate ≤ -3, so the robot is only allowed to move in the region with x-coordinate between -3 and ∞. The input to the constraint network is 18-dimensional state space, including positional and angular values of different body parts of the cheetah, followed by their velocities, making constraint extraction challenging due to this high-dimensional space.

We collected 30 demonstrations and run PUCL to learn constraint. Since each dimension of the state space represents distinct physical properties and scales, we identify the reliable infeasible data based on the standardized dataset as discussed in Section III-B. Euclidean distance is used as the distance measure[4] with a threshold[5] $d_r = 2.6$. The learning performance is presented in Fig. 5, where PUCL achieves the highest IoU metrics and the second best unsafe rate. This confirms the ability of PUCL for learning constraints in high-dimensional state spaces.

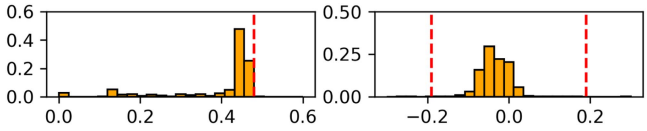


Fig. 6. Velocity distribution of the policy learned with PUCL (Left: v_x, Right: v_z). The horizontal axis represents the velocity, and the red line represents the true constraint threshold.

TABLE II
COMPARISON OF TWO POLICY REPRESENTATION AND LEARNING METHODS

Methods	CRL-PUCL	DSM-PUCL
IoU	0.46 ± 0.09	0.42 ± 0.01
Recall rate	$(94.2 \pm 4.5)\%$	(99.9 ± 0.1) %
Precision	$(47.6 \pm 10.0)\%$	(42.1 ± 1.3) %
Unsafe rate	$(0.47 \pm 0.82)\%$	$(0.15 \pm 0.56)\%$
Training time (min)	13.4 ± 1.7	4.7 ± 0.4

B. Comparison of Policy Representation and Learning Methods

As introduced in Section III-D, we studied two policy representation and learning approaches, namely constrained RL and DSM. We test the two variants with PUCL as the constraint learning method in the 3D reaching task. The two frameworks are named CRL-PUCL and DSM-PUCL, respectively, and the results of comparison are presented in Table II.[6] Training was conducted on a PC with a 12th Intel i9-12900K $\times$ 24 and an

[4]We also tested Manhattan distance and Chebyshev distance. Both achieves similar performance with appropriately selected distance thresholds.

[5]This value is tuned following the discussion in III-B for heterogeneous cases: we first collect $\mathcal{P}$ using an unconstrained policy and compute the score $\mathcal{S}_{\mathcal{D}}(\bar{s})$ of each state from $\mathcal{P}$. We hypothesis that around 80% of states in $\mathcal{P}$ are reliable infeasible data, and this corresponds to a threshold of 2.6.

[6]The recall rate is computed as (True infeasible)/(True infeasible + False feasible), and precision is (True infeasible)/(True infeasible + False infeasible)

Authorized licensed use limited to: EPFL LAUSANNE. Downloaded on October 26,2026 at 09:32:26 UTC from IEEE Xplore. Restrictions apply.

NVIDIA GeForce RTX 3070. All results are the average of 15 independent runs.

DSM-PUCL and CRL-PUCL achieve similar classification performance in IoU, recall rate and precision, and DSM-PUCL generally exhibits lower variance. For the learned policy, while both methods have an unsafe rate below 1%, the rate of DSM-PUCL is slightly lower than CRL-PUCL. This is because DSM-PUCL modulates the policy in a principled way and features a safety guarantee [19]. DSM-PUCL also requires less training time than CRL-PUCL. Based on these comparisons, we conclude that both methods are effective for learning constraints. For robotic tasks where the control variable is the velocity/acceleration of each state dimension, DSM may be more favourable than CRL in terms of training time and policy performance, while CRL is more suitable for more general task with more complex environment dynamics and rewards.

C. Constraint Transfer to Variant of the Same Task

One remarkable advantage of learning constraint is that the learned constraint can be transferred to relevant task with the same constraint but potentially different goals and rewards. To demonstrate this, we consider the task of robot avoiding four cups forming two sets of complex shapes (see Fig. 1). We collected demonstrations using DSM and applied DSM-PUCL to learn this constraint. Then the learned constraint network is transferred to a variant task with a set of shifted goal states and starting points. The new nominal policy heading to new target is directly modulated with the transferred constraint with DSM. The modulated policy also avoids these cups despite the change of task configuration.

V. Conclusions and Limitations

This letter proposed a two-step Positive-Unlabeled Constraint Learning method to infer an arbitrary, potentially nonlinear, constraint function from demonstration. The proposed method treats the demonstration as positive data and the higher-reward policy as unlabeled data. It uncovers the infeasible states by first identifying reliable infeasible data using a distance metric, and training a feasibility classifier with the identified data. The benefits of the proposed method were demonstrated by learning a 2D and 3D position constraint, a 3D velocity constraint and a constraint in high-dimensional environment, using either a constrained RL policy or a DSM policy. It managed to recover the constraints and outperformed other baseline methods in terms of accuracy and safety.

Although the proposed method is effective, its performance depends on the choice of an appropriate distance threshold and the specific distance metric. In the future, we aim to address this limitation by exploring methods to jointly learn a distance representation that can automatically adapt to various tasks.

References

[1] J. Garcia and F. Fernández, "A comprehensive survey on safe reinforcement learning," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 1437–1480, 2015.

[2] R. Noothigattu et al., "Teaching AI agents ethical values using reinforcement learning and policy orchestration," *IBM J. Res. Dev.*, vol. 63, no. 4/5, pp. 2–1, 2019.

[3] G. Chou, D. Berenson, and N. Ozay, "Learning constraints from demonstrations," in *Proc. 13th Workshop Algorithmic Found. Robot.*, 2020, pp. 228–245.

[4] D. R. Scobee and S. S. Sastry, "Maximum likelihood constraint inference for inverse reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2020, doi: 10.48550/arXiv.2206.09670.

[5] A. Glazier, A. Loreggia, N. Mattei, T. Rahgooy, F. Rossi, and K. B. Venable, "Learning behavioral soft constraints from demonstrations," in *Proc. Workshop Safe Robust Contr. Uncertain Syst. NeurIPS 2021*, 2021.

[6] D. L. McPherson, K. C. Stocking, and S. S. Sastry, "Maximum likelihood constraint inference from stochastic demonstrations," in *Proc. 2021 IEEE Conf. Control Technol. Appl.*, 2021, pp. 1208–1213.

[7] K. C. Stocking, D. L. McPherson, R. P. Matthew, and C. J. Tomlin, "Maximum likelihood constraint inference on continuous state spaces," in *Proc. 2022 Int. Conf. Robot. Automat.*, 2022, pp. 8598–8604.

[8] S. Malik, U. Anwar, A. Aghasi, and A. Ahmed, "Inverse constrained reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 7390–7399.

[9] G. Liu, Y. Luo, A. Gaurav, K. Rezaee, and P. Poupart, "Benchmarking constraint inference in inverse reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2023, doi: 10.48550/arXiv.2206.09670.

[10] J. Jang, M. Song, and D. Park, "Inverse constraint learning and generalization by transferable reward decomposition," *IEEE Robot. Autom. Letters*, vol. 9, no. 1, pp. 279–286, 2023.

[11] G. Chou, N. Ozay, and D. Berenson, "Learning constraints from locally-optimal demonstrations under cost function uncertainty," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 3682–3690, Apr. 2020.

[12] G. Chou, H. Wang, and D. Berenson, "Gaussian process constraint learning for scalable chance-constrained motion planning from demonstrations," *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 3827–3834, Apr. 2022.

[13] G. Chou, N. Ozay, and D. Berenson, "Learning parametric constraints in high dimensions from demonstrations," in *Proc. Conf. Robot Learn.*, 2020, pp. 1211–1230.

[14] J. Bekker and J. Davis, "Learning from positive and unlabeled data: A survey," *Mach. Learn.*, vol. 109, pp. 719–760, 2020.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[16] B. Zhang and W. Zuo, "Reliable negative extracting based on knn for learning from positive and unlabeled examples," *J. Comput.*, vol. 4, no. 1, pp. 94–101, 2009.

[17] L. Liu and T. Peng, "Clustering-based method for positive and unlabeled text categorization enhanced by improved TFIDF," *J. Inf. Sci. Eng.*, vol. 30, no. 5, pp. 1463–1481, 2014.

[18] A. Stooke, J. Achiam, and P. Abbeel, "Responsive safety in reinforcement learning by PID Lagrangian methods," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 9133–9143.

[19] L. Huber, A. Billard, and J.-J. Slotine, "Avoidance of convex and concave obstacles with convergence ensured through contraction," *IEEE Robot. Automat. Lett.*, vol. 4, no. 2, pp. 1462–1469, Apr. 2019.

[20] B. Peng et al., "Model-based chance-constrained reinforcement learning via separated proportional-integral Lagrangian," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 1, pp. 466–478, Jan. 2024.

[21] B. Peng, Y. Mu, J. Duan, Y. Guan, S. E. Li, and J. Chen, "Separated proportional-integral Lagrangian for chance constrained reinforcement learning," in *Proc. 2021 IEEE Intell. Veh. Symp.*, 2021, pp. 193–199.

[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[23] J. Rey, K. Kronander, F. Farshidian, J. Buchli, and A. Billard, "Learning motions from demonstrations and rewards with time-invariant dynamical systems based policies," *Auton. Robots*, vol. 42, pp. 45–64, 2018.

Authorized licensed use limited to: EPFL LAUSANNE. Downloaded on October 26,2026 at 09:32:26 UTC from IEEE Xplore. Restrictions apply.