Learning from Human Directional Corrections

Wanxin Jin, Todd D. Murphey, Zehui Lu, Shaoshuai Mou

***Abstract*—This paper proposes a novel approach that enables a robot to learn an objective function incrementally from human directional corrections. Existing methods learn from human magnitude corrections; since a human needs to carefully choose the magnitude of each correction, those methods can easily lead to over-corrections and learning inefficiency. The proposed method only requires human directional corrections — corrections that only indicate the direction of an input change without indicating its magnitude. We only assume that each correction, regardless of its magnitude, points in a direction that improves the robot's current motion relative to an unknown objective function. The allowable corrections satisfying this assumption account for half of the input space, as opposed to the magnitude corrections which have to lie in a shrinking level set. For each directional correction, the proposed method updates the estimate of the objective function based on a cutting plane method, which has a geometric interpretation. We have established theoretical results to show the convergence of the learning process. The proposed method has been tested in numerical examples, a user study on two human-robot games, and a real-world quadrotor experiment. The results confirm the convergence of the proposed method and further show that the method is significantly more effective (higher success rate), efficient/effortless (less human corrections needed), and potentially more accessible (fewer early wasted trials) than the state-of-the-art robot learning frameworks.**

***Index Terms*—Learning from corrections, human-robot physical interaction, motion planning, cutting-plane method, inverse reinforcement learning, learning from demonstrations.**

I. Introduction

FOR tasks where robots work in proximity to human users, a robot is required to not only guarantee the accomplishment of the task but also complete it in a way that a human user prefers. Different users may have different preferences about how the robot should perform the task. Such customized requirements usually lead to a considerable workload of robot programming, which requires human users to have high expertise and skills.

To circumvent the above limitations of robot programming, learning from demonstrations (LfD) empowers non-expert users to program robots by providing demonstrations. In LfD [1], a user first provides a robot with demonstrations in a *one-time* manner, then the robot learns a control policy or objective function *offline* from the demonstrations. While achieving the notable success in various applications [2]–[4], the *one-time* and *offline* nature of LfD could lead to some challenges. For example, when the demonstration data is insufficient to infer an objective function due to the low informativeness [5] or deviation from optima [6], the demonstrations have to be re-collected and the robot has to be re-trained. This is particularly inconvenient for robots with high degrees of freedom.

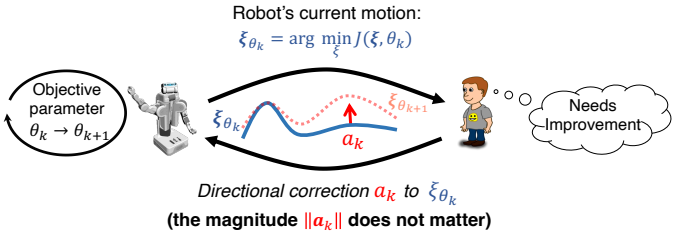


Fig. 1: An illustration of learning from directional corrections proposed in this paper. The robot's current motion $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ optimizes an objective function parameterized by $\boldsymbol{\theta}_k$. A human user applies a directional correction $\boldsymbol{a}_k$ to $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$. Note that the magnitude $\|\boldsymbol{a}_k\|$ of this correction does not matter. After receiving $\boldsymbol{a}_k$, the robot updates its objective function parameter to $\boldsymbol{\theta}_{k+1}$. This motion-correction-update process repeats until the convergence of the objective function parameter.

A recent line of work [6]–[9] addresses the above challenges of LfD by developing a scheme that enables a robot to learn an objective function from *user's feedback or corrections*. Fig. 1 is an example of those learning schemes. At each iteration, a human user does not need to provide optimal demonstrations to the robot, but merely a correction which is an *incremental improvement* to the robot's current motion. The robot then leverages the correction to update its objective function. Compared to LfD, this incremental learning scheme reduces the workload of a user, especially for those (such as novices) who cannot provide the optimal demonstrations in a one-time manner [10]. Despite its promise, the state-of-the-art methods [6]–[9] still face some challenges as below.

First, providing a valid correction—the one that improves the robot motion—can be nontrivial. To obtain a valid correction, the human user must carefully choose the *magnitude* of a correction. As a robot gets closer to the 'desired motion' (i.e., the motion in the human's perspective), the allowable range of the magnitude of valid corrections gets smaller (see Section II for the detailed explanation). Thus, the human can easily over-correct a robot near the desired motion, i.e., giving too large corrections that adversely drive the robot away from the

Wanxin Jin is with the General Robotics, Automation, Sensing and Perception (GRASP) Laboratory, University of Pennsylvania, PA 19104, USA. Wanxin Jin is the corresponding author. Email: wanxinjin@gmail.com.

Todd D. Murphey is with the Department of Mechanical Engineering, Northwestern University, Evanston, IL 60208, USA. This material is partially based upon work supported by the National Science Foundation under award 1837515. Email: t-murphey@northwestern.edu.

Zehui Lu and Shaoshuai Mou are with the School of Aeronautics and Astronautics, Purdue University, West Lafayette, IN 47906, USA. Dr. Mou's research is supported in part by grants from the Research in Applications for Learning Machines (REALM) Consortium of Northrop Grumman Corporation, Rolls-Royce Corporation, and the NASA University Leadership Initiative (ULI). Email: zehuilu789@gmail.com, mous@purdue.edu.

This work involved human subjects or animals in its research. Approval of all ethical and experimental procedures and protocols was granted by Purdue's Institutional Review Board under Application No. IRB-2021-1539, and performed in the line with Title 45 of the Code of Federal Regulations, Part 46 (45CFR 46).

desired motion. Such difficulty makes robot learning inefficient (we will experimentally show this in Sections V-C and VI).

Second, most of existing methods [7]–[9] lack *theoretical guarantee* for the learning performance. While [6] shows that learning from human magnitude corrections can attain a bounded regret [11], this regret bound still cannot explicitly tell whether or not the learned objective function converges to the true one induced by all corrections.

To address the above challenges, this paper proposes a new learning method that enables a robot to learn an objective function from human *directional* corrections with *theoretical convergence guarantee*. We highlight the following new features of the proposed method, as shown by Fig. 1.

(I) The proposed learning scheme only requires the user's *directional corrections*. A directional correction is a correction that only concerns directional information and does not necessarily need to be magnitude-specific. For instance, in teaching a mobile robot, a directional correction can be simply as 'go left' or 'go right' without dictating how far the robot should go. Furthermore, unlike existing work [6]–[9], the proposed approach can *directly* handle the sparse directional corrections without a pre-processing step of creating a human 'intended' trajectory, which could introduce artifacts and unfavorably affect the learning performance.

(II) We emphasize the theoretical foundations of the proposed method. First, the learning process is based on cutting-plane methods and thus has an intuitive geometric interpretation; and second, we establish the theoretical guarantees to show the convergence of our method towards finding the true objective function induced by human directional corrections.

We have conducted extensive experiments to test the proposed method, including various numerical examples, a user study on two human-robot games, and a real-world quadrotor experiment. The results show that the proposed method is significantly more effective (higher success rate), efficient/effortless (less human corrections needed), and potentially more accessible (fewer early wasted trials) than the state-of-the-art robot learning frameworks.

A. Related Work

1) One-time Learning from Demonstrations: To learn an objective function from demonstrations, routine methods are inverse optimal control [12]–[14] or inverse reinforcement learning [15]–[17], where an objective function is learned from optimal demonstrations and subsequently used for control or planning. Successful LfD applications include autonomous driving [2], robot manipulation [3], and motion planning [4]. Despite the notable success [18]–[20], LfD could be inconvenient in practice. First, demonstrations in LfD are usually collected in a *one-time* manner and the robot learning is *offline*. In the case where demonstration data is insufficient to recover an objective function, such as low data informativeness as discussed in [5], or significantly deviating from the optima, the data has to be re-collected and the training has to be re-run. Second, existing LfD [13]–[17] normally requires the optimality of demonstrations, which are challenging to collect for robots with high degree-of-freedoms. For example, when providing demonstrations for a humanoid robot, a user has to account for the robot's motion of all degrees of freedom in a spatially and temporally consistent manner [6].

2) Incrementally Learning from Corrections/feedback: Compared to one-time LfD, learning from corrections or feedback enables a user to *incrementally* improve robot motion, making it more suitable for non-expert users who cannot provide optimal demonstrations in a one-time fashion [10]. The key assumption is that robot motion after correction achieves a higher reward (or a lower cost) than before correction. Under this assumption, [6] develops a co-active learning method to update the robot objective function using user feedback. The user feedback includes a human either selecting the top-ranked robot trajectory among all candidates or physically demonstrating a preferred trajectory to the robot. By defining a regret, which quantifies the average misalignment between the value of robot motion and that of the desired motion under the true objective function, [6] shows the convergence of the regret. But since regret only considers values of an objective function, it cannot directly tell and guarantee that the learned objective function itself is converging towards the true one. As we will show in Section V-C, the objective function can converge to a local solution instead of a true one.

Recently, [7]–[9] handle learning from corrections through the perspective of the partially observable Markov decision process (POMDP). Here, human corrections are viewed as observations about the unknown objective function parameter. By approximating the observation model and applying the maximum a posteriori estimation, they obtain a learning formula similar to the co-active learning [6]. Along with this perspective, some variants have been recently developed to particularly account for the uncertainties of human corrections. For example, [21] simultaneously estimates a rationality coefficient to characterize the rationality confidence of human corrections. [9] fits the inverse reinforcement learning into a Kalman filter framework to quantitatively capture the uncertainty of the estimation.

Both the above co-active learning and POMDP-based learning require a human to carefully choose the *magnitude* correction that improves robot motion. As we will detail in Section II, choosing a valid magnitude correction is difficult, especially when a robot gets closer to the desired motion, because the allowable magnitude of a correction has to lie in a shrinking level set. Near the desired motion, a human could easily over-correct the robot, i.e., applying too large corrections that adversely drive the robot away from the desired motion, making the algorithm diverge or even fail. We will experimentally show this difficulty in Section V-C. Moreover, due to the sparsity of human corrections, all the above methods require a dedicated step to obtain a *human 'intended' trajectory* for each correction. In the co-active learning, a robot needs to switch to a screening mode to obtain a human preferred trajectory, and in the POMDP-based method, a human intended trajectory is created by trajectory deformation [22]. These pre-processing steps may not only introduce more hyperparameters but also add some artifacts that could undermine the learning performance [8]

Very recently, [23] directly learns a desired trajectory from

human physical interactions, then a robot tracks the learned trajectory using linear–quadratic regulator (LQR) controllers. They minimize a trajectory distance loss, and the learning update is based on the assumption that the direction of a human correction is exactly aligned with the deepest gradient descent of the distance loss. While that work is related to our work in the sense that they both use the direction of a human correction, our work, however, focuses on learning an objective function instead of learning the desired trajectory directly. Importantly, we do not restrict the direction of a correction to be exactly aligned with the deepest gradient descent, but any correction as long as it has a direction that decreases the cost of robot motion. The above two distinctions lead to a new method of our work, which has a strong theoretical guarantee for learning convergence.

Finally, we want to mention that most of the existing methods [7]–[9], [23] lack theoretical guarantee about the learning performance. The only work that attempts to do so is [6], where a regret bound is shown. As discussed previously, such regret is an indicator of the misalignment of the values of the objective function, and cannot directly tell or guarantee the convergence of the objective function itself. In this paper, we will directly characterize the convergence of the learned objective function towards the true function that is induced by human corrections.

B. Contributions

To give readers a high-level picture of the proposed method, we show an overview in Fig. 2. We claim the following contributions of the proposed method.

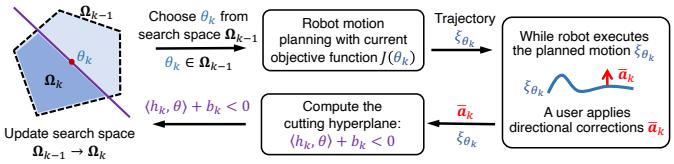


Fig. 2: An overview of the proposed algorithm.

(I) The proposed method learns an objective function from human *directional corrections*. It only requires that a correction, regardless of its magnitude, points in a direction of improving robot motion. As we will show in Sections II and V-C, the allowable corrections that satisfy this requirement always account for half of the input space, making it more flexible for users to choose corrections from. Also, unlike existing methods, the proposed method directly learns from sparse directional corrections without a *pre-processing* step of creating an 'intended' trajectory.

(II) The proposed learning algorithm has strong theoretical foundations. First, the proposed learning method is based on the cutting plane technique, which has straightforward geometric interpretations, as shown in Fig. 2. Second, we have established the theoretical results to show the convergence of the method towards finding the true objective function induced by all corrections.

Numerical examples, a user study on two human-robot games, and a real-world quadrotor experiment confirm the efficacy and convergence of the proposed algorithm. The user study and real-world experiment further show that the proposed method is significantly more effective (higher success rate), efficient/effortless (fewer corrections needed), and potentially more accessible (fewer early wasted trials) than the state-of-the-art methods.

In what follows, Section II describes the problem formulation. Section III outlines the main algorithm and presents its geometric interpretation. Section IV provides the theoretical results of the algorithm and its detailed implementation. Various numerical examples and comparisons are given in Section V. Section VI presents a user study and Section VII provides a real-world experiment. Conclusions are drawn in Section VIII. The appendix includes the proof, discussion, and possible extension of this work.

II. Problem Formulation

Consider a robot with the following dynamics and the initial condition:

$$\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t), \quad \text{with} \quad \boldsymbol{x}_0, \tag{1}$$

where $\boldsymbol{x}_t \in \mathbb{R}^n$ is the robot state, $\boldsymbol{u}_t \in \mathbb{R}^m$ is the control input, $\boldsymbol{f} : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$ is differentiable, and $t = 1, 2, \cdots$ is the time step. As commonly used in objective learning methods [5]–[10], [14]–[17], suppose that the robot cost function has the following parameterized form:

$$J(\boldsymbol{u}_{0:T}, \boldsymbol{\theta}) = \sum\nolimits_{t=0}^{T} \boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{u}_t) + h(\boldsymbol{x}_{T+1}), \tag{2}$$

where $\boldsymbol{\phi} : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^r$ is a vector of the *pre-defined* features (basis functions); $\boldsymbol{\theta} \in \mathbb{R}^r$ is a vector of weights, which are *tunable*; and $h(\boldsymbol{x}_{T+1})$ is the final cost on the robot final state $\boldsymbol{x}_{T+1}$, such as penalizing the distance between $\boldsymbol{x}_{T+1}$ and a given goal state $\boldsymbol{x}^{\text{goal}}$, $h(\boldsymbol{x}_{T+1}) = \|\boldsymbol{x}^{\text{goal}} - \boldsymbol{x}_{T+1}\|^2$. Note that depending on specific applications, the final cost term $h(\boldsymbol{x}_{T+1})$ can be either absent, i.e., $h(\boldsymbol{x}_{T+1}) = 0$, or also parameterized in a weighted-feature form with the weights to be learned jointly with $\boldsymbol{\theta}$. The method developed in this paper can sufficiently handle either cases with little modifications. For a fixed choice of $\boldsymbol{\theta}$, the robot plans a sequence of inputs $\boldsymbol{u}_{0:T}$ over time horizon T by (locally) optimizing the cost function (2) subject to (1), producing a trajectory

$$\boldsymbol{\xi}_{\boldsymbol{\theta}} = \left\{ \boldsymbol{x}^{\boldsymbol{\theta}}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}}_{0:T} \right\}. \tag{3}$$

In the following text, we occasionally write the cost function (2) as $J(\boldsymbol{\theta})$ for ease of readability.

For a specific task, suppose that a human user expects the robot to minimize an *implicit* cost function $J(\boldsymbol{\theta}^*)$ in the same form of (2) but with *unknown* $\boldsymbol{\theta}^*$. We call $\boldsymbol{\theta}^*$ the *true weight vector*. In general, a human user may neither explicitly write down the value of $\boldsymbol{\theta}^*$ nor demonstrate the corresponding *desired trajectory* $\boldsymbol{\xi}_{\boldsymbol{\theta}^*}$, but the human user can tell whether the robot motion trajectory is *desired* or not. A robot trajectory is desired if it minimizes the implicit $J(\boldsymbol{\theta}^*)$; otherwise, it is not desired. In order for the robot to learn $J(\boldsymbol{\theta}^*)$ (and thus generate the desired trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}^*}$), the human user is only able to make corrections to the robot motion. After receiving each human correction, the robot updates its guess $\boldsymbol{\theta}$ towards the true $\boldsymbol{\theta}^*$. This procedure has been shown in Fig. 1.

Specifically, the process of a robot learning from human directional corrections is iterative, as shown in Fig. 2. Each iteration basically includes three steps: robot planning (& execution), human correction, and robot update. Let $k = 1, 2, 3, \cdots$, be the iteration index and let $\boldsymbol{\theta}_k$ denote the robot's guess of the weight vector at iteration k. At $k = 1$, the robot is initialized with an arbitrary guess $\boldsymbol{\theta}_1$. At iteration k, the robot first performs motion *planning*, i.e. solving $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ by minimizing the cost function $J(\boldsymbol{\theta}_k)$ in (2) subject to its dynamics in (1). During the robot executing $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$, the human user observes $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ and applies a *correction*, denoted by $\boldsymbol{a}_{t_k} \in \mathbb{R}^m$, to the robot in its input space. Here, $t_k \in \{0, 1, \cdots, T\}$ is called *correction time*, indicating at which time step the correction $\boldsymbol{a}_{t_k}$ is made. After receiving $\boldsymbol{a}_{t_k}$, the robot *updates* its guess $\boldsymbol{\theta}_k$ to $\boldsymbol{\theta}_{k+1}$ based on an update rule developed later.

One distinguishing feature of our method is that we assume that each human correction $\boldsymbol{a}_{t_k}$ satisfies

$$\left\langle -\nabla J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*),\ \boldsymbol{a}_k \right\rangle > 0, \quad k = 1, 2, 3, \cdots. \tag{4}$$

Here,

$$\boldsymbol{a}_k = \left[\mathbf{0}^\mathsf{T} \quad \cdots, \quad \boldsymbol{a}^\mathsf{T}_{t_k}, \quad \cdots, \mathbf{0}^\mathsf{T}\right]^\mathsf{T} \in \mathbb{R}^{m(T+1)}, \tag{5}$$

with $\boldsymbol{a}_{t_k}$ filled at the t_kth entry and $\mathbf{0} \in \mathbb{R}^m$ elsewhere. Here, $\langle \cdot, \cdot \rangle$ is the dot product, and $-\nabla J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*)$ is the gradient-descent of $J(\boldsymbol{\theta}^*)$ with respect to $\boldsymbol{u}_{0:T}$ evaluated at the robot current trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k} = \{\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}\}$. Note that (4) does not require a specific value to the magnitude of $\boldsymbol{a}_{t_k}$ but requires its direction roughly around the gradient-descent direction of $J(\boldsymbol{\theta}^*)$. It means that the correction $\boldsymbol{a}_k$ aims to guide the robot motion $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ towards a lower cost under $J(\boldsymbol{\theta}^*)$ unless the trajectory is desired. Thus, we call $\boldsymbol{a}_{t_k}$ a *directional correction*.

In practice, one always needs to account for human's noisy or imperfect corrections. Thus, we modify (4) as follows

$$\begin{aligned} &\mathbb{E}_{\boldsymbol{a}_k}\left\langle -\nabla J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*),\ \boldsymbol{a}_k \right\rangle \\ &= \left\langle -\nabla J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*),\ \mathbb{E}_{\boldsymbol{a}_k}(\boldsymbol{a}_k) \right\rangle \qquad (6a) \\ &= \left\langle -\nabla J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*),\ \bar{\boldsymbol{a}}_k \right\rangle > 0, \quad k = 1, 2, ..., \end{aligned}$$

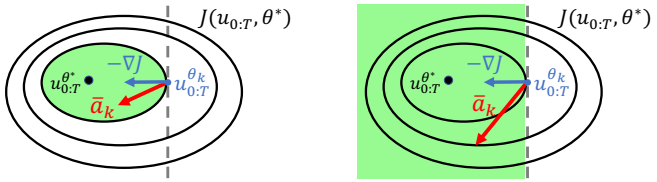
where

$$\bar{\boldsymbol{a}}_k = \mathbb{E}_{\boldsymbol{a}_k}(\boldsymbol{a}_k). \tag{6b}$$

Here, $\mathbb{E}(\cdot)$ is the expectation with respect to the human correction data $\boldsymbol{a}_k$. (6) says that the original assumption (4) can be met in expectation; in other words, the robot can receive and average human's *multiple* directional corrections before one update of the cost function is made. There are two ways to implement multiple corrections. The most convenient way is that a human user applies multiple directional corrections at different time steps but all within the same execution of the robot motion $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$, i.e., the robot executes the current plan $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ just once. The second way is that the multiple corrections are applied in the multiple repetitions of the execution of $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$, i.e., the robot executes the same plan $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ for multiple times. Whichever implementation, by taking the expectation of human's multiple directional corrections $\bar{\boldsymbol{a}}_k = \mathbb{E}_{\boldsymbol{a}_k}(\boldsymbol{a}_k)$, (6) permits noisy and imperfect human directional corrections. The similar strategies have also been adopted in [6].

The **problem of interest** in this paper is that given human (averaged) directional corrections $\bar{\boldsymbol{a}}_k$ satisfying the assumption (6), we aim to develop a rule to update the robot weight vector guess $\boldsymbol{\theta}_k$ such that $\boldsymbol{\theta}_k$ converges to $\boldsymbol{\theta}^*$ as $k = 1, 2, 3, ...$

Remark 1. *We assume that human corrections $\boldsymbol{a}_{t_k} \in \mathbb{R}^m$ are in the robot input space, which means that $\boldsymbol{a}_{t_k}$ can be directly added to robot input $\boldsymbol{u}_{t_k}$. This can be readily fulfilled in some applications such as autonomous vehicles, where a user directly changes the steering angle of a vehicle. For other cases where the corrections are not readily in the robot input space, it could be met via some specific interfaces (or computation), which map correction signals to the robot input space. For example, when a human interacts with the end effector of a robot manipulator, one can convert the task-space contact force to the joint torques via Jacobian. Then, $\boldsymbol{a}_{t_k}$ is the mapped correction. The reason why we do not consider the corrections in the state space is that 1) corrections in input spaces may be easier to implement, as shown in our later experiments, and 2) corrections in state spaces can be infeasible for the under-actuated robot systems [24].*



(a) Allowable region (green) of magnitude corrections.

(b) Allowable region (green) of directional corrections.

Fig. 3: Magnitude corrections v.s. directional corrections. Contours (circles) and the desired trajectory $\boldsymbol{u}^{\boldsymbol{\theta}^*}_{0:T}$ (black dot) of the true cost function $J(\boldsymbol{\theta}^*)$ are plotted. (a) Green region (a sub-level set) indicates all allowable magnitude corrections $\bar{\boldsymbol{a}}_k$ that satisfy $J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T} + \bar{\boldsymbol{a}}_k, \boldsymbol{\theta}^*) < J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*)$. (b) Green region (half of the input space) indicates all allowable directional corrections $\bar{\boldsymbol{a}}_k$ that satisfy $\langle -\nabla J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*), \bar{\boldsymbol{a}}_k \rangle > 0$.

Remark 2. *Assumption (4) on directional correction $\boldsymbol{a}_{t_k}$ is less restrictive than ones in [6]–[9] using magnitude corrections, which assume the cost of the corrected robot trajectory $\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T} + \bar{\boldsymbol{a}}_k$ is lower than that of the robot original motion $\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}$, i.e., $J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T} + \bar{\boldsymbol{a}}_k, \boldsymbol{\theta}^*) < J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*)$. As shown in Fig. 3, their assumptions usually lead to the restriction of correction magnitude. Specifically, to satisfy $J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T} + \bar{\boldsymbol{a}}_k, \boldsymbol{\theta}^*) < J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*)$, $\|\bar{\boldsymbol{a}}_k\|$ has to be chosen from the $J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*)$-sublevel set of $J(\boldsymbol{\theta}^*)$, as shown by the green region in Fig. 3a. Furthermore, this region will shrink as $\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}$ gets close to the desired trajectory $\boldsymbol{u}^{\boldsymbol{\theta}^*}_{0:T}$ (black dot), thus making $\|\bar{\boldsymbol{a}}_k\|$ more difficult to choose. In contrast, the directional corrections satisfying (4) always account for half of the input space, as shown by the green region in Fig. 3b. A human user can choose any correction as long as its direction lies in the half space of the gradient descent of $J(\boldsymbol{\theta}^*)$. Thus, (4) would be more likely to be satisfied for non-expert users. We will experimentally show this advantage later in Section V-C and Section VI.*

III. Main Algorithm Outline and its Geometric Interpretation

In this section, we will outline the proposed main algorithm and present its geometric interpretation.

A. Hyperplane for Each Directional Correction

Before developing the main algorithm, we first show that the condition (6) equals a linear inequality imposed on the true $\boldsymbol{\theta}^*$. This has been formally stated by the following lemma.

Lemma 1. *Suppose that the current guess of the weight vector is $\boldsymbol{\theta}_k$, and the robot trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}=\{\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}\}$ is a result of (locally) minimizing $J(\boldsymbol{\theta}_k)$ in (2) subject to dynamics (1). For $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$, given a human (averaged) directional correction $\bar{\boldsymbol{a}}_{t_k}$ satisfying (6), one has*

$$\langle \boldsymbol{h}_k, \boldsymbol{\theta}^* \rangle + b_k < 0, \quad k = 1, 2, 3 \cdots, \tag{7}$$

with

$$\boldsymbol{h}_k = \boldsymbol{H}_1^{\mathsf{T}}(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})\bar{\boldsymbol{a}}_k \ \in \mathbb{R}^r, \tag{8a}$$

$$b_k = \bar{\boldsymbol{a}}_k^{\mathsf{T}}\boldsymbol{H}_2(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})\nabla h(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{T+1}) \ \in \mathbb{R}. \tag{8b}$$

Here, $\bar{\boldsymbol{a}}_k$ is defined in (6b); $\nabla h(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{T+1})$ is the gradient of the final cost $h(\boldsymbol{x}_{T+1})$ in (2) evaluated at $\boldsymbol{x}^{\boldsymbol{\theta}_k}_{T+1}$; $\boldsymbol{H}_1(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})$ and $\boldsymbol{H}_2(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})$ are computed as:

$$\boldsymbol{H}_1(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}) = \begin{bmatrix} \boldsymbol{F}_u\boldsymbol{F}_x^{-1}\boldsymbol{\Phi}_x + \boldsymbol{\Phi}_u \\ \frac{\partial \boldsymbol{\phi}^{\mathsf{T}}}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}} \end{bmatrix} \in \mathbb{R}^{m(T+1)\times r}, \tag{9a}$$

$$\boldsymbol{H}_2(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}) = \begin{bmatrix} \boldsymbol{F}_u\boldsymbol{F}_x^{-1}\boldsymbol{V} \\ \frac{\partial \boldsymbol{f}^{\mathsf{T}}}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}} \end{bmatrix} \in \mathbb{R}^{m(T+1)\times n}, \tag{9b}$$

with

$$\boldsymbol{F}_x = \begin{bmatrix} I & \frac{-\partial \boldsymbol{f}^{\mathsf{T}}}{\partial \boldsymbol{x}_1^{\boldsymbol{\theta}_k}} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & I & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & I & \frac{-\partial \boldsymbol{f}^{\mathsf{T}}}{\partial \boldsymbol{x}_{T-1}^{\boldsymbol{\theta}_k}} \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & & I \end{bmatrix}, \ \boldsymbol{\Phi}_x = \begin{bmatrix} \frac{\partial \boldsymbol{\phi}^{\mathsf{T}}}{\partial \boldsymbol{x}_1^{\boldsymbol{\theta}_k}} \\ \frac{\partial \boldsymbol{\phi}^{\mathsf{T}}}{\partial \boldsymbol{x}_2^{\boldsymbol{\theta}_k}} \\ \vdots \\ \frac{\partial \boldsymbol{\phi}^{\mathsf{T}}}{\partial \boldsymbol{x}_T^{\boldsymbol{\theta}_k}} \end{bmatrix}, \tag{10a}$$

$$\boldsymbol{F}_u = \begin{bmatrix} \frac{\partial \boldsymbol{f}^{\mathsf{T}}}{\partial \boldsymbol{u}_0^{\boldsymbol{\theta}_k}} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \frac{\partial \boldsymbol{f}^{\mathsf{T}}}{\partial \boldsymbol{u}_1^{\boldsymbol{\theta}_k}} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \frac{\partial \boldsymbol{f}^{\mathsf{T}}}{\partial \boldsymbol{u}_{T-1}^{\boldsymbol{\theta}_k}} \end{bmatrix}, \ \boldsymbol{\Phi}_u = \begin{bmatrix} \frac{\partial \boldsymbol{\phi}^{\mathsf{T}}}{\partial \boldsymbol{u}_0^{\boldsymbol{\theta}_k}} \\ \frac{\partial \boldsymbol{\phi}^{\mathsf{T}}}{\partial \boldsymbol{u}_1^{\boldsymbol{\theta}_k}} \\ \vdots \\ \frac{\partial \boldsymbol{\phi}^{\mathsf{T}}}{\partial \boldsymbol{u}_{T-1}^{\boldsymbol{\theta}_k}} \end{bmatrix}, \tag{10b}$$

$$\boldsymbol{V} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}_T^{\boldsymbol{\theta}_k}} \end{bmatrix}^{\mathsf{T}}. \tag{10c}$$

In above, the dimensions are $\boldsymbol{F}_x \in \mathbb{R}^{nT\times nT}$, $\boldsymbol{F}_u \in \mathbb{R}^{mT\times nT}$, $\boldsymbol{\Phi}_x \in \mathbb{R}^{nT\times r}$, $\boldsymbol{\Phi}_u \in \mathbb{R}^{mT\times r}$, $\boldsymbol{V} \in \mathbb{R}^{nT\times n}$. I is $n\times n$ identity. For a general differentiable function $\boldsymbol{g}(\boldsymbol{x})$ and a fixed value $\boldsymbol{x}^*$, we denote $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}^*}$ as the Jacobian of $\boldsymbol{g}(\boldsymbol{x})$ evaluated at $\boldsymbol{x}^*$.

Proof. We first derive the explicit form of $\nabla J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*)$ in (6), and then show that (6) can be re-written as (7).

Consider the robot current trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}=\{\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}\}$, which satisfies the robot dynamics constraint (1). For any $t = 0, 1, ..., T$, define the infinitesimal perturbation pair $(\delta\boldsymbol{x}_t, \delta\boldsymbol{u}_t)$ at the state-input pair $(\boldsymbol{x}_t^{\boldsymbol{\theta}_k}, \boldsymbol{u}_t^{\boldsymbol{\theta}_k})$. By linearizing the dynamics (1) around $(\boldsymbol{x}_t^{\boldsymbol{\theta}_k}, \boldsymbol{u}_t^{\boldsymbol{\theta}_k})$, we have

$$\delta\boldsymbol{x}_{t+1} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}_t^{\boldsymbol{\theta}_k}}\delta\boldsymbol{x}_t + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}_t^{\boldsymbol{\theta}_k}}\delta\boldsymbol{u}_t, \tag{11}$$

where $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}_t^{\boldsymbol{\theta}_k}}$ and $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}_t^{\boldsymbol{\theta}_k}}$ are the Jacobians of $\boldsymbol{f}$ with respect to $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$, respectively, evaluated at $(\boldsymbol{x}_t^{\boldsymbol{\theta}_k}, \boldsymbol{u}_t^{\boldsymbol{\theta}_k})$. By stacking (11) for all $t = 0, 1, ..., T$ and also noting $\delta\boldsymbol{x}_0 = \boldsymbol{0}$ (because $\boldsymbol{x}_0^{\boldsymbol{\theta}_k} = \boldsymbol{x}_0$ is given), we have the following compact form:

$$-\boldsymbol{A}^{\mathsf{T}}\delta\boldsymbol{x}_{1:T+1} + \boldsymbol{B}^{\mathsf{T}}\delta\boldsymbol{u}_{0:T} = \boldsymbol{0}, \tag{12}$$

with $\delta\boldsymbol{x}_{1:T+1}=[\delta\boldsymbol{x}_1^{\mathsf{T}}, \cdots, \delta\boldsymbol{x}_{T+1}^{\mathsf{T}}]^{\mathsf{T}}$, $\delta\boldsymbol{u}_{1:T}=[\delta\boldsymbol{u}_1^{\mathsf{T}}, \cdots, \delta\boldsymbol{u}_T^{\mathsf{T}}]^{\mathsf{T}}$,

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{F}_x & -\boldsymbol{V} \\ \boldsymbol{0} & I \end{bmatrix}, \quad \text{and} \quad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{F}_u & \boldsymbol{0} \\ \boldsymbol{0} & \frac{\partial \boldsymbol{f}^{\mathsf{T}}}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}} \end{bmatrix}, \tag{13}$$

with $\boldsymbol{F}_x$ and $\boldsymbol{F}_u$ defined in (10). Given $\delta\boldsymbol{x}_{1:T+1}$ and $\delta\boldsymbol{u}_{1:T}$, the increment of $J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*)$, denoted as $\delta J(\boldsymbol{\theta}^*)$, is

$$\delta J(\boldsymbol{\theta}^*) = \boldsymbol{C}^{\mathsf{T}}\delta\boldsymbol{x}_{1:T+1} + \boldsymbol{D}^{\mathsf{T}}\delta\boldsymbol{u}_{0:T}, \tag{14}$$

with

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{\Phi}_x\boldsymbol{\theta}^* \\ \frac{\partial h^{\mathsf{T}}}{\partial \boldsymbol{x}_{T+1}^{\boldsymbol{\theta}_k}} \end{bmatrix} \quad \text{and} \quad \boldsymbol{D} = \begin{bmatrix} \boldsymbol{\Phi}_u\boldsymbol{\theta}^* \\ \frac{\partial \boldsymbol{\phi}^{\mathsf{T}}}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}}\boldsymbol{\theta}^* \end{bmatrix}, \tag{15}$$

with $\boldsymbol{\Phi}_x$ and $\boldsymbol{\Phi}_u$ defined in (10). Since $\boldsymbol{A}$ is always invertible (due to full rank), we solve for $\delta\boldsymbol{x}_{1:T+1}$ from (12) and submit it to (14), yielding

$$\begin{aligned} \delta J(\boldsymbol{\theta}^*) &= \boldsymbol{C}^{\mathsf{T}}\delta\boldsymbol{x}_{1:T+1} + \boldsymbol{D}^{\mathsf{T}}\delta\boldsymbol{u}_{0:T}, \\ &= \Big(\boldsymbol{C}^{\mathsf{T}}(\boldsymbol{A}^{-1})^{\mathsf{T}}\boldsymbol{B}^{\mathsf{T}} + \boldsymbol{D}^{\mathsf{T}}\Big)\delta\boldsymbol{u}_{0:T}. \end{aligned} \tag{16}$$

Thus, we have

$$\nabla J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*) = \boldsymbol{B}\boldsymbol{A}^{-1}\boldsymbol{C} + \boldsymbol{D}. \tag{17}$$

The above (17) can be further written as

$$\begin{aligned} &\nabla J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*) = \boldsymbol{B}\boldsymbol{A}^{-1}\boldsymbol{C} + \boldsymbol{D} \\ &= \begin{bmatrix} \boldsymbol{F}_u & \boldsymbol{0} \\ \boldsymbol{0} & \frac{\partial \boldsymbol{f}^{\mathsf{T}}}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}} \end{bmatrix} \begin{bmatrix} \boldsymbol{F}_x & -\boldsymbol{V} \\ \boldsymbol{0} & I \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\Phi}_x\boldsymbol{\theta}^* \\ \frac{\partial h^{\mathsf{T}}}{\partial \boldsymbol{x}_{T+1}^{\boldsymbol{\theta}_k}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\Phi}_u\boldsymbol{\theta}^* \\ \frac{\partial \boldsymbol{\phi}^{\mathsf{T}}}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}}\boldsymbol{\theta}^* \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{F}_u\boldsymbol{F}_x^{-1}\boldsymbol{\Phi}_x\boldsymbol{\theta}^* + \boldsymbol{\Phi}_u\boldsymbol{\theta}^* + \boldsymbol{F}_u\boldsymbol{F}_x^{-1}\boldsymbol{V}\frac{\partial h^{\mathsf{T}}}{\partial \boldsymbol{x}_{T+1}^{\boldsymbol{\theta}_k}} \\ \frac{\partial \boldsymbol{\phi}^{\mathsf{T}}}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}}\boldsymbol{\theta}^* + \frac{\partial \boldsymbol{f}^{\mathsf{T}}}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}}\frac{\partial h^{\mathsf{T}}}{\partial \boldsymbol{x}_{T+1}^{\boldsymbol{\theta}_k}} \end{bmatrix}, \end{aligned} \tag{18}$$

where we have used Schur complement to compute the inverse of the block matrix $\boldsymbol{A}$. By the definitions in (9), (18) becomes

$$\begin{aligned} \nabla J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*) &= \boldsymbol{H}_1(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})\boldsymbol{\theta}^* \\ &\quad + \boldsymbol{H}_2(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})\nabla h(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{T+1}). \end{aligned} \tag{19}$$

Substituting (19) into (6) and also considering the definitions in (8), we obtain

$$\Big\langle -\nabla J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*), \ \bar{\boldsymbol{a}}_k \Big\rangle = -\langle \boldsymbol{h}_k, \boldsymbol{\theta}^* \rangle - b_k > 0, \tag{20}$$

which leads to (7). This completes the proof. □

We have the following remarks on the intuition of Lemma 1.

Remark 3. *The key result of Lemma 1 is the linear inequality (7), which is associated with a hyperplane* $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k = 0$. *The parameters* $\boldsymbol{h}_k$ *and* b_k *of this hyperplane, given in (8), are both known and determined by human directional correction* $\bar{\boldsymbol{a}}_k$ *and robot current trajectory* $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$. *In other words, Lemma 1 states that given robot motion* $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ *and the human directional correction* $\bar{\boldsymbol{a}}_k$, *one can write a hyperplane* $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k = 0$ *and a linear inequality (7) for the unknown* $\boldsymbol{\theta}^*$.

Remark 4. *Obtaining* $\boldsymbol{h}_k$ *and* b_k *requires computing the matrices* $\boldsymbol{H}_1(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})$ *and* $\boldsymbol{H}_2(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})$ *in (9). In our previous work [5],* $\boldsymbol{H}_1(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})$ *and* $\boldsymbol{H}_2(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})$ *are called the 'recovery matrix'. The Lemma 1 of [5] has shown that the 'recovery matrix' can be iteratively computed by integrating each point* $(\boldsymbol{x}^{\boldsymbol{\theta}_k}_t, \boldsymbol{u}^{\boldsymbol{\theta}_k}_t)$, $t = 0, 1, ..., T$. *This iterative property of the recovery matrix facilitates the computation of* $\boldsymbol{H}_1$ *and* $\boldsymbol{H}_2$ *by avoiding the inverse of* $\boldsymbol{F}_x$ *in (9), significantly reducing the computational burden. Please refer to Lemma 1 of [5] for the iteration formula.*

B. Outline of the Proposed Algorithm

With Lemma 1, we are ready to develop the main algorithm to learn $\boldsymbol{\theta}^*$. At each iteration k, we maintain a *weight search space*, denoted by $\boldsymbol{\Omega}_k \subset \mathbb{R}^r$, such that $\boldsymbol{\theta}^* \in \boldsymbol{\Omega}_k$ and $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_k$ for all $k = 1, 2, 3, ...$ This $\boldsymbol{\Omega}_k$ can be thought of as the possible location of $\boldsymbol{\theta}^*$, and $\boldsymbol{\theta}_k$ as the current guess to $\boldsymbol{\theta}^*$. Rather than a direct rule to guide $\boldsymbol{\theta}_k$ towards $\boldsymbol{\theta}^*$, we will develop a rule to update $\boldsymbol{\Omega}_k$ to $\boldsymbol{\Omega}_{k+1}$ such that a useful measure of the size of $\boldsymbol{\Omega}_k$ will converge to 0. With this high-level idea, we outline the proposed main algorithm below.

Main Algorithm (Outline)

Initialize the weight search space $\boldsymbol{\Omega}_0$ *to be*

$$\boldsymbol{\Omega}_0 = \{\boldsymbol{\theta} \in \mathbb{R}^r \mid -\underline{c}_i \leq \boldsymbol{\theta}[i] \leq \bar{c}_i,\ i = 1, 2, ..., r\}, \quad (21)$$

where constants $\underline{c}_i \geq 0$ *and* $\bar{c}_i \geq 0$ *denote the lower bound and upper bounds of the* i*th entry* $\boldsymbol{\theta}[i]$ *in* $\boldsymbol{\theta}$, *respectively. Here,* $\underline{c}_i$ *and* $\bar{c}_i$ *can be chosen large enough such that* $\boldsymbol{\theta}^* \in \boldsymbol{\Omega}_0$. *At iteration* $k = 1, 2, ...,$ *the learning proceeds with the following three steps:*

Step 1: *Choose a weight vector guess* $\boldsymbol{\theta}_k$ *from the current weight search space* $\boldsymbol{\Omega}_{k-1}$, *i.e.,* $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$*(we will discuss how to choose* $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$ *in Section IV).*

Step 2: *The robot restarts and plans its motion trajectory* $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ *by solving a trajectory optimization with the cost function* $J(\boldsymbol{\theta}_k)$ *in (2) and the dynamics in (1). While the robot is executing the plan* $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$, *a human applies (averaged) directional correction* $\bar{\boldsymbol{a}}_k$ *in (6b). Then, a hyperplane* $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k = 0$ *is computed in (7)-(8).*

Step 3: *Update the weight search space* $\boldsymbol{\Omega}_{k-1}$ *to* $\boldsymbol{\Omega}_k$ *via*

$$\boldsymbol{\Omega}_k = \boldsymbol{\Omega}_{k-1} \cap \{\boldsymbol{\theta} \in \mathbb{R}^r \mid \langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k < 0\}. \quad (22)$$

We provide some remarks to the above outline of the proposed algorithm. For the initialization (21), we allow different entries of $\boldsymbol{\theta}$ to have their own lower and upper bounds, which may come from prior knowledge. Simply but not necessarily, one could also initialize

$$\boldsymbol{\Omega}_0 = \{\boldsymbol{\theta} \in \mathbb{R}^r \mid \|\boldsymbol{\theta}\|_\infty \leq R\}, \quad (23)$$

where

$$R = \max\{\underline{c}_i,\ \bar{c}_i,\ i = 1, \cdots, r\}. \quad (24)$$

In Step 1, a weight vector guess $\boldsymbol{\theta}_k$ is chosen from $\boldsymbol{\Omega}_{k-1}$. We will (in (26) in Lemma 2) show that the true $\boldsymbol{\theta}^*$ *always* lies in $\boldsymbol{\Omega}_k$ for any $k = 1, 2, 3, ...$ Thus, we will expect $\boldsymbol{\theta}_k$ to be closer to $\boldsymbol{\theta}^*$ if a size measure of $\boldsymbol{\Omega}_k$ gets smaller. In fact, the weight search space $\boldsymbol{\Omega}_k$ is *always non-increasing* as $\boldsymbol{\Omega}_k \subseteq \boldsymbol{\Omega}_{k-1}$ due to (22) in Step 3. In the next Section IV, we will further focus on how to pick $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$ such that the size of $\boldsymbol{\Omega}_k$ is *strictly reduced* as k increases. In Step 2, the robot trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ is planned by solving a trajectory optimization with the cost function $J(\boldsymbol{\theta}_k)$ in (2) and the dynamics constraint (1). This can be done by any available trajectory optimization methods, e.g., [25] and existing solvers [26]. After a human applies a directional correction $\bar{\boldsymbol{a}}_{t_k}$ to the executed $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$, a hyperplane $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k = 0$ can be computed via (7)-(8), which will be used to update the search space by (22) in Step 3. The detailed implementation of the above main algorithm will be described in the next section.

Given the above proposed main algorithm, we next present the following important lemma.

Lemma 2. *Under the proposed main algorithm, one has*

$$\langle \boldsymbol{h}_k, \boldsymbol{\theta}_k \rangle + b_k = 0, \quad \forall\ k = 1, 2, 3, ... \quad (25)$$

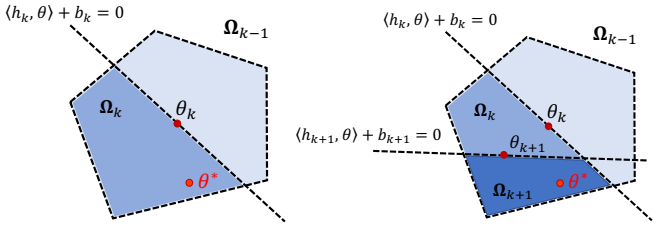
and

$$\boldsymbol{\theta}^* \in \boldsymbol{\Omega}_k, \quad \forall\ k = 1, 2, 3, ... \quad (26)$$

A proof of Lemma 2 is given in Appendix A. Lemma 2 has the following intuitive geometric explanations. First, (25) says that the current guess $\boldsymbol{\theta}_k$ is always located on the current hyperplane $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k = 0$. Second, (26) says that although the proposed algorithm directly updates the weight search space $\boldsymbol{\Omega}_k$ via (22), the true (but unknown) weight vector $\boldsymbol{\theta}^*$ is always contained in the current $\boldsymbol{\Omega}_k$. Thus, intuitively, the smaller the search space $\boldsymbol{\Omega}_k$ is, the closer $\boldsymbol{\theta}^*$ is to $\boldsymbol{\theta}_k$.

C. Geometric Interpretation

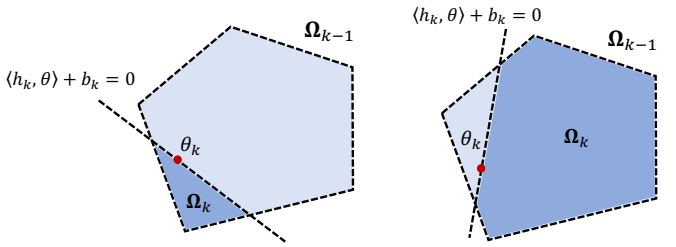
In this part, we will provide an interpretation of the proposed main algorithm through a geometric perspective. For simplicity of illustrations, we assume $\boldsymbol{\theta} \in \mathbb{R}^2$ here.



(a) At k-th iteration (b) At $(k+1)$-th iteration

Fig. 4: Illustration of updating $\boldsymbol{\Omega}_k$.

At the kth iteration shown in Fig. 4a, by Step 1 of the main algorithm, a weight vector guess $\boldsymbol{\theta}_k$ (red dot) is picked from the current search space $\boldsymbol{\Omega}_{k-1}$ (light blue region), i.e., $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$. By Step 2, we obtain a hyperplane $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k = 0$ (black dashed line), which cuts through the weight search space $\boldsymbol{\Omega}_{k-1}$ into two portions. By (25) of Lemma 2, we know that $\boldsymbol{\theta}_k$ always lies on this hyperplane because $\langle \boldsymbol{h}_k, \boldsymbol{\theta}_k \rangle + b_k = 0$, as shown in Fig. 4a. By Step 3 of the main algorithm, we only keep one of the two cut portions, which is the interaction between $\boldsymbol{\Omega}_{k-1}$ and the half space $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k < 0$, and the kept portion will be used as the new search space for the next iteration, that is, $\boldsymbol{\Omega}_k = \boldsymbol{\Omega}_{k-1} \cap \{\boldsymbol{\theta} \in \mathbb{R}^r \mid \langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k < 0\}$, as shown by the blue region in Fig. 4a. The above procedure repeats at the next iteration $k+1$ in Fig. 4b, and finally produces a smaller search space $\boldsymbol{\Omega}_{k+1}$, as shown by the darkest blue region in Fig. 4b. From (22), one has $\boldsymbol{\Omega}_0 \supseteq \cdots \boldsymbol{\Omega}_{k-1} \supseteq \boldsymbol{\Omega}_k \supseteq \boldsymbol{\Omega}_{k+1} \supseteq \cdots$. Moreover, by (26) in Lemma 2, we note that the true $\boldsymbol{\theta}^*$ (red point) is always contained in $\boldsymbol{\Omega}_k$ whenever k is.



(a) A large cut from $\boldsymbol{\Omega}_{k-1}$ (b) A small cut from $\boldsymbol{\Omega}_{k-1}$

Fig. 5: Illustration of how different human directional corrections $\bar{\boldsymbol{a}}_k$ affect the cut of the weight search space $\boldsymbol{\Omega}_{k-1}$.

Besides the above geometric illustration, we also have the following observations:

(1) The key idea of the proposed main algorithm is to cut and remove the weight search space $\boldsymbol{\Omega}_{k-1}$ as each human directional correction $\bar{\boldsymbol{a}}_k$ becomes available. Thus, we always expect that $\boldsymbol{\Omega}_{k-1}$ can quickly shrink as k increases, because thereby we can say that the guess $\boldsymbol{\theta}_k$ is close to the true weight vector $\boldsymbol{\theta}^*$. As shown in Fig. 4, the reduction rate of $\boldsymbol{\Omega}_{k-1}$ depends on two factors: the human directional correction $\bar{\boldsymbol{a}}_k$, and how to choose $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$, both discussed below.

(2) From (8), we note that human directional correction $\bar{\boldsymbol{a}}_k$ determines $\boldsymbol{h}_k$, which is the normal vector of the hyperplane $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k = 0$. When fixing $\boldsymbol{\theta}_k$, we can think of the hyperplane rotates around $\boldsymbol{\theta}_k$ with different $\bar{\boldsymbol{a}}_k$, leading to different removals of $\boldsymbol{\Omega}_{k-1}$. This can be illustrated by comparing Fig. 5a and Fig. 5b.

(3) The choice of the guess $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$ defines the specific location of the hyperplane $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k = 0$, because the hyperplane is always passing through $\boldsymbol{\theta}_k$ by (25) in Lemma 2. Thus, $\boldsymbol{\theta}_k$ also affects how $\boldsymbol{\Omega}_{k-1}$ is cut and removed. This can be illustrated by comparing Fig. 4a with Fig. 5a.

Based on the above observations, we know that the convergence of the proposed main algorithm is determined by the reduction of the weight search space $\boldsymbol{\Omega}_{k-1}$. This reduction depends on both the human directional correction $\bar{\boldsymbol{a}}_k$ and the choice of the weight vector guess $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$. In the next section, we will present a way of choosing $\boldsymbol{\theta}_k$ to guarantee the convergence of the proposed main algorithm.

IV. Algorithm Implementation and Convergence Analysis

In this section, we detail the choice of $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$, show the convergence of the proposed algorithm, and present a detailed implementation of the algorithm with a termination criterion.

A. Choice of Current Guess $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$

In the proposed main algorithm, at iteration k, the weight search space $\boldsymbol{\Omega}_{k-1}$ is updated using (22), rewritten below:

$$\boldsymbol{\Omega}_k = \boldsymbol{\Omega}_{k-1} \cap \{\boldsymbol{\theta} \in \mathbb{R}^r \mid \langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k < 0\}.$$

To show the reduction of a weight search space, it is straightforward to use the volume measure (length or area for one- or two-dimension cases, respectively) of a (closure) search space $\boldsymbol{\Omega}_k$, denoted as $\mathbf{Vol}(\boldsymbol{\Omega}_k)$. Zero volume implies the convergence of the search space [27]. By (22), we know that $\boldsymbol{\Omega}_k \subseteq \boldsymbol{\Omega}_{k-1}$ and thus $\mathbf{Vol}(\boldsymbol{\Omega}_k)$ is non-increasing. In the following we need to further find a way such that $\mathbf{Vol}(\boldsymbol{\Omega}_k)$ is strictly decreasing, i.e., there exists a constant $0 \leq \alpha < 1$ such that

$$\mathbf{Vol}(\boldsymbol{\Omega}_k) \leq \alpha \, \mathbf{Vol}(\boldsymbol{\Omega}_{k-1}). \tag{27}$$

From Fig. 5, we observe that for a fixed choice of $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$, different human directional corrections $\bar{\boldsymbol{a}}_k$ can lead to different reduction of $\boldsymbol{\Omega}_{k-1}$. For example, Fig. 5a has a large volume reduction from $\boldsymbol{\Omega}_{k-1}$ to $\boldsymbol{\Omega}_k$ while Fig. 5b leads to a very small reduction. Unfortunately, we cannot assume the specific direction of a human correction $\bar{\boldsymbol{a}}_k$ (it is the discretion of the user), but we can choose the current guess $\boldsymbol{\theta}_k$ 'smartly' to avoid the very small reduction regardless of specific human corrections. One intuitive choice of such $\boldsymbol{\theta}_k$ is at some *center* of the search space $\boldsymbol{\Omega}_{k-1}$. Thus, we define the following center of the maximum volume ellipsoid (MVE) inscribed in $\boldsymbol{\Omega}_{k-1}$.

Definition 1 (Maximum Volume Inscribed Ellipsoid [28])**.** *Given a compact convex set $\boldsymbol{\Omega} \subset \mathbb{R}^r$, the maximum volume ellipsoid (MVE) inscribed in $\boldsymbol{\Omega}$, defined as $\boldsymbol{E}$, is denoted by*

$$\boldsymbol{E} = \{\bar{B}\boldsymbol{\theta} + \bar{\boldsymbol{d}} \mid \|\boldsymbol{\theta}\|_2 \leq 1\}. \tag{28}$$

Here, $\bar{B} \in \mathbb{S}^r_{++}$ (i.e., a $r \times r$ positive definite matrix) and $\bar{\boldsymbol{d}} \in \mathbb{R}^r$ is called the center of $\boldsymbol{E}$. $\bar{B}$ and $\bar{\boldsymbol{d}}$ solve the optimization:

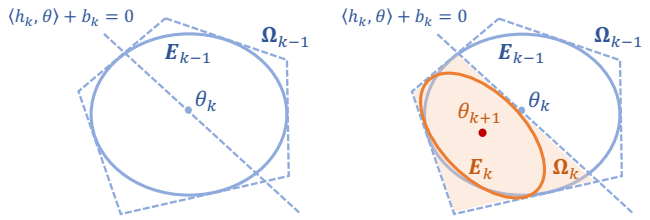
$$\begin{aligned} \max_{\boldsymbol{d}, B \in \mathbb{S}^r_{++}} \quad & \log\det B \\ s.t. \quad & \sup_{\|\boldsymbol{\theta}\|_2 \leq 1} \mathrm{I}_{\boldsymbol{\Omega}}(B\boldsymbol{\theta} + \boldsymbol{d}) \leq 0, \end{aligned} \tag{29}$$

where $\mathrm{I}_{\boldsymbol{\Omega}}(\boldsymbol{\theta}) = 0$ for $\boldsymbol{\theta} \in \boldsymbol{\Omega}$ and $\mathrm{I}_{\boldsymbol{\Omega}}(\boldsymbol{\theta}) = \infty$ for $\boldsymbol{\theta} \notin \boldsymbol{\Omega}$.

By Definition 1, let $\boldsymbol{E}_k$ denote the MVE inscribed in $\boldsymbol{\Omega}_k$ and $\boldsymbol{d}_k$ denote the center of $\boldsymbol{E}_k$, $k = 0, 1, \ldots$. For the choice of $\boldsymbol{\theta}_{k+1} \in \boldsymbol{\Omega}_k$, we choose

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{d}_k, \tag{30}$$

as illustrated in Fig. 6. Other choices of $\boldsymbol{\theta}_{k+1}$ using other types of centers of the search space are discussed in Appendix C.



(a) Center of MVE in $\boldsymbol{\Omega}_{k-1}$ (b) Center of MVE in $\boldsymbol{\Omega}_k$

Fig. 6: Illustration of choosing $\boldsymbol{\theta}_{k+1}$ as the center of MVE $\boldsymbol{E}_k$ inscribed in the weight search space $\boldsymbol{\Omega}_k$.

We now give a computational method to solve for $\boldsymbol{d}_k$, i.e. the center of MVE $\boldsymbol{E}_k$ inscribed in $\boldsymbol{\Omega}_k$. Recall that in the main algorithm, the initial $\boldsymbol{\Omega}_0$ is (21), which is equivalent to

$$\boldsymbol{\Omega}_0 = \left\{ \boldsymbol{\theta} \;\middle|\; \begin{array}{c} \langle \boldsymbol{e}_i, \boldsymbol{\theta}\rangle - \bar{c}_i \le 0 \\ -\langle \boldsymbol{e}_i, \boldsymbol{\theta}\rangle - \underline{c}_i \le 0 \end{array}, \quad i = 1, \cdots, r \right\}, \tag{31}$$

where $\boldsymbol{e}_i$ is a unit vector with the only ith entry as 1. Following the update (22), $\boldsymbol{\Omega}_k$ is also a compact polytope, which is

$$\boldsymbol{\Omega}_k = \left\{ \boldsymbol{\theta} \;\middle|\; \begin{array}{l} \langle \boldsymbol{e}_i, \boldsymbol{\theta}\rangle - \bar{c}_i \le 0, \; i = 1, \cdots, r; \\ -\langle \boldsymbol{e}_i, \boldsymbol{\theta}\rangle - \underline{c}_i \le 0, \; i = 1, \cdots, r; \\ \langle \boldsymbol{h}_j, \boldsymbol{\theta}\rangle + b_j < 0, \; j = 1, \cdots, k \end{array} \right\}. \tag{32}$$

As a result, solving (29) for the center $\boldsymbol{d}_k$ becomes a convex program [28], stated in the lemma below.

Lemma 3. *For a polytope $\boldsymbol{\Omega}_k$ in (32), the center $\boldsymbol{d}_k$ of the MVE $\boldsymbol{E}_k$ inscribed in $\boldsymbol{\Omega}_k$ can be solved via the following convex optimization:*

$$\begin{aligned} \min_{\boldsymbol{d}, B \in \mathbb{S}^r_{++}} \; & -\log\det B \\ s.t. \; & \|B\boldsymbol{e}_i\|_2 + \langle \boldsymbol{d}, \boldsymbol{e}_i\rangle \le \bar{c}_i, \; i{=}1, \cdots r, \\ & \|B\boldsymbol{e}_i\|_2 - \langle \boldsymbol{d}, \boldsymbol{e}_i\rangle \le \underline{c}_i, \; i{=}1, \cdots r, \\ & \|B\boldsymbol{h}_j\|_2 + \langle \boldsymbol{d}, \boldsymbol{h}_j\rangle \le -b_j, \; j{=}1, \cdots k. \end{aligned} \tag{33}$$

The proof of Lemma 3 can be found in Chapter 8.4.2 in [28, pp.414]. (33) can be efficiently solved by available convex program solvers, e.g., [29]. In the implementation, since the number of linear inequalities grows as k increases, the trick of dropping some redundant inequalities in (32) can be adopted [27]. Dropping redundant inequalities does not change $\boldsymbol{\Omega}_k$ and its volume reduction (convergence). Please see how to identify the redundant inequalities in [27].

B. Exponential Convergence and Termination Criterion

Now we analyze the convergence of the volume reduction of $\boldsymbol{\Omega}_k$ and develop its termination criterion in implementation. First, the convergence of the reduction of $\textbf{Vol}(\boldsymbol{\Omega}_k)$ is guaranteed by the following lemma.

Lemma 4. *Let $\boldsymbol{\theta}_k$ be chosen as the center $\boldsymbol{d}_{k-1}$ of the MVE $\boldsymbol{E}_{k-1}$ inscribed in $\boldsymbol{\Omega}_{k-1} \subset \mathbb{R}^r$. Then, the update (22) has*

$$\frac{\boldsymbol{Vol}(\boldsymbol{\Omega}_k)}{\boldsymbol{Vol}(\boldsymbol{\Omega}_{k-1})} \le (1 - \frac{1}{r}). \tag{34}$$

Lemma 4 is a theorem directly from [30]. Lemma 4 indicates

$$\textbf{Vol}\,(\boldsymbol{\Omega}_k) \le (1 - \frac{1}{r})^k \,\textbf{Vol}\,(\boldsymbol{\Omega}_0).$$

Thus, the convergence rate of $\textbf{Vol}\,(\boldsymbol{\Omega}_k) \to 0$ is as fast as $(1 - \frac{1}{r})^k \to 0$, as $k = 0, 1, 2, ...$

To implement the proposed main algorithm, we will not only need the exponential convergence as established by Lemma 4 but also a termination criterion, which specifies the maximum number of iterations needed for $\textbf{Vol}\,(\boldsymbol{\Omega}_k)$ below certain a given threshold. Thus, we have the following theorem.

Theorem 1. *In the main algorithm, suppose $\boldsymbol{\Omega}_0$ is given by (21), and at iteration k, $\boldsymbol{\theta}_k$ is chosen as the center $\boldsymbol{d}_{k-1}$ of the MVE $\boldsymbol{E}_{k-1}$ inscribed in $\boldsymbol{\Omega}_{k-1}$. Given a termination condition*

$$\boldsymbol{Vol}\,(\boldsymbol{\Omega}_k) \le (2\epsilon)^r$$

with ϵ a user-specified threshold, the proposed main algorithm runs for $k \le K$ iterations, namely, the algorithm terminates in at most K iterations, where

$$K = \frac{r\log(R/\epsilon)}{-\log(1 - 1/r)}, \tag{35}$$

with R given in (24).

Proof. Initially, we have $\textbf{Vol}\,(\boldsymbol{\Omega}_0) \le (2R)^r$. From Lemma 4, after k iterations, we have

$$\textbf{Vol}\,(\boldsymbol{\Omega}_k) \le (1 - \frac{1}{r})^k \,\textbf{Vol}\,(\boldsymbol{\Omega}_0) \le (1 - \frac{1}{r})^k (2R)^r, \tag{36}$$

which yields to

$$\log \textbf{Vol}\,(\boldsymbol{\Omega}_k) \le k\log(1 - \frac{1}{r}) + \log(2R)^r. \tag{37}$$

When $k = \frac{r\log(R/\epsilon)}{-\log(1-1/r)}$,

$$\log \textbf{Vol}\,(\boldsymbol{\Omega}_k) \le -r\log(R/\epsilon) + \log(2R)^r. \tag{38}$$

The above equation is simplified to

$$\log \textbf{Vol}\,(\boldsymbol{\Omega}_k) \le \log(2\epsilon)^r, \tag{39}$$

which means that the termination condition $\textbf{Vol}\,(\boldsymbol{\Omega}_k) \le (2\epsilon)^r$ is satisfied. This completes the proof. □

We have the following comments on the above Theorem 1.

Remark 5. *Since Lemma 2 states that both $\boldsymbol{\theta}^*$ and $\boldsymbol{\theta}_k$ are in $\boldsymbol{\Omega}_k$ for any $k = 1, 2, 3, ...$, the user-specified threshold ϵ in the termination condition $\boldsymbol{Vol}\,(\boldsymbol{\Omega}_k) \le (2\epsilon)^r$ can also be understood as an indicator for the distance between the true $\boldsymbol{\theta}^*$ (usually unknown in practice) and the robot current guess $\boldsymbol{\theta}_k$. ϵ is set based on the desired learning accuracy.* [1]

C. Detailed Implementation of Main Algorithm

With the termination criterion stated in Theorem 1 and the choice of $\boldsymbol{\theta}_k$ in (30) and Lemma 3, we present the detailed implementation of the proposed main method in Algorithm 1. The setting of the initial $\boldsymbol{\Omega}_0$ in (21) and ϵ will be given in Appendix B-D.

[1] In practice, it is usually easy to set ϵ with a small value, because the empirical results in Sections VI and VII show that the robot motion trajectory can converge (to the desired motion) more quickly than the objective function itself. This means that one usually sees the convergence of the robot trajectory before the termination condition is reached, as shown in Fig. 10. More discussion about setting ϵ will be given in Appendix B-D.

Algorithm 1: Learning from directional corrections

Input: Specify a termination threshold ϵ and use it to compute the maximum iteration K by (35).

Initialization: Initial weight search space $\boldsymbol{\Omega}_0$ in (21).

for $k = 1, 2, \cdots, K$ **do**

- Choose a weight vector guess $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$ via Lemma 3;
- Restart and plan a robot trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ by solving a trajectory optimization with the cost function $J(\boldsymbol{\theta}_k)$ in (2) and the dynamics in (1);
- Robot executes the planned trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ while receving the human averaged directional correction $\bar{\boldsymbol{a}}_k$ in (6b);
- Compute the matrices $\boldsymbol{H}_1(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})$ and $\boldsymbol{H}_2(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})$, and then compute the hyperplane and half space $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k < 0$ via (7)-(8);
- Update the weight search space by $\boldsymbol{\Omega}_k = \boldsymbol{\Omega}_{k-1} \cap \{\boldsymbol{\theta} \in \mathbb{R}^r \mid \langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k < 0\}$ via (22);

end

Output: $\boldsymbol{\theta}_K$.

V. Numerical Examples

In this section, we will test the proposed method in numerical simulations and make comparisons with the magnitude-correction methods discussed in the previous related work.

A. Inverted Pendulum

The dynamics of a pendulum is

$$\ddot{\alpha} = \frac{-g}{l}\sin\alpha - \frac{d}{ml^2}\dot{\alpha} + \frac{u}{ml^2}, \tag{40}$$

with α being the angle between the pendulum and the direction of gravity, u is the torque applied to the pivot, $l = 1$m, $m = 1$kg, and $d = 0.1$Ns/m are the length, mass, and damping ratio of the pendulum, respectively. We discretize the dynamics using the Euler method with a time interval $\Delta = 0.2$s. The state and control input of the pendulum are $\boldsymbol{x} = [\alpha, \dot{\alpha}]^\mathsf{T}$ and $\boldsymbol{u} = u$, respectively. The initial condition is $\boldsymbol{x}_0 = [0, 0]^\mathsf{T}$. In the cost function (2), we set the feature and weight vectors as

$$\begin{aligned} \boldsymbol{\phi} &= [\alpha^2, \alpha, \dot{\alpha}^2, u^2]^\mathsf{T} \in \mathbb{R}^4, \\ \boldsymbol{\theta} &= [\theta_1, \theta_2, \theta_3, \theta_4]^\mathsf{T} \in \mathbb{R}^4, \end{aligned} \tag{41}$$

respectively, and the final cost $h(\boldsymbol{x}_{T+1}) = 10(\alpha-\pi)^2 + 10\dot{\alpha}^2$, as our goal is to swing up the pendulum. The discrete time horizon is $T = 30$.

For numerical analysis, we 'simulate' the human directional corrections instead of using real human corrections. The simulated corrections are generated as follows. Suppose that the true weight vector is known: $\boldsymbol{\theta}^* = [0.5, 0.5, 0.5, 0.5]^\mathsf{T}$. At iteration k, a simulated correction $\boldsymbol{a}_{t_k}$ is generated using the *sign* of the gradient of the true cost function $J(\boldsymbol{\theta}^*)$, i.e.,

$$\boldsymbol{a}_{t_k} = -\text{sign}\left(\nabla J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*)\,[t_k]\right) \in \mathbb{R}. \tag{42}$$

Here, $\nabla J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*)\,[t_k]$ denotes the t_kth entry of ∇J, and the correction time t_k is randomly drawn from the a uniform distribution over horizon $[0, T]$. Obviously, the above simulated directional corrections satisfies the assumption (4).

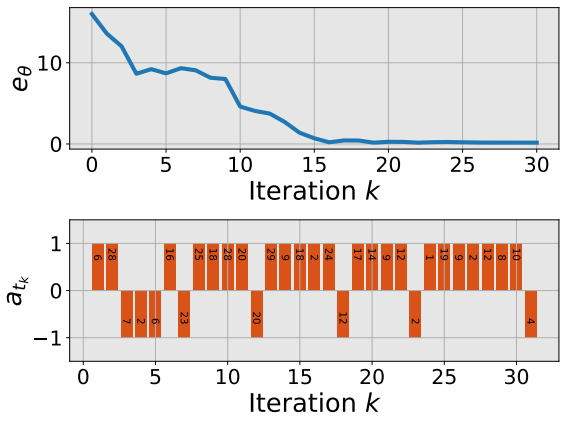


Fig. 7: Learning from simulated directional corrections for the pendulum. The upper panel shows the weight error $e_{\boldsymbol{\theta}} = \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2$ versus iteration. The bottom shows the simulated directional correction $\boldsymbol{a}_{t_k}$ (positive or negative sign) at each iteration k; here, the value in each bar is t_k, which is randomly drawn from a uniform distribution over horizon $[0, 30]$.

The initial weight search space is set as $\boldsymbol{\Omega}_0 = \{\boldsymbol{\theta} \,|\, 0 \le \boldsymbol{\theta}[i] \le 5,\ i = 1, 2, 3, 4\}$. In Algorithm 1, we set the termination threshold $\epsilon = 10^{-1}$, and the maximum learning iteration solved via (35) is $K = 55$. We apply Algorithm 1 to learn the true $\boldsymbol{\theta}^*$. To illustrate results, we define the weight error $e_{\boldsymbol{\theta}} = \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2$ and plot $e_{\boldsymbol{\theta}}$ versus iteration k in the top panel of Fig. 7. In the bottom panel of Fig. 7, we plot the simulated directional correction $\boldsymbol{a}_{t_k}$ at each iteration k, where $+1$ and -1 bar denote the positive and negative sign (i.e., direction) of correction $\boldsymbol{a}_{t_k}$ in (42), respectively; the number inside the bar denotes the correction time t_k randomly drawn from $[0, T]$.

From the results in Fig. 7, we can see that as the learning iteration k increases, the weight vector guess $\boldsymbol{\theta}_k$ converges to the true $\boldsymbol{\theta}^* = [0.5, 0.5, 0.5, 0.5]^\mathsf{T}$. This shows the efficacy of the method, as guaranteed by Theorem 1.

B. Two-link Robot Arm System

Here, we test the proposed method on a two-link robot arm. The dynamics of the robot arm system (moving horizontally) is $M(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau}$, where $M(\boldsymbol{q})$ is the inertia matrix, $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is the Coriolis and centrifugal term; $\boldsymbol{q} = [q_1, q_2]^\mathsf{T}$ is the vector of joint angles, and $\boldsymbol{\tau} = [\tau_1, \tau_2]^\mathsf{T}$ is the vector of joint toques. The state and control input are $\boldsymbol{x} = [\boldsymbol{q}, \dot{\boldsymbol{q}}]^\mathsf{T} \in \mathbb{R}^4$ and $\boldsymbol{u} = \boldsymbol{\tau} \in \mathbb{R}^2$, respectively. The initial condition of the robot arm is set as $\boldsymbol{x}_0 = [0, 0, 0, 0]^\mathsf{T}$. All physical parameters in the dynamics are units. We discretize the dynamics using the Euler method with a time interval $\Delta = 0.2$s. In the cost function (2), we set the feature and weight vectors as

$$\boldsymbol{\phi} = [q_1^2,\ q_1,\ q_2^2,\ q_2,\ \|\boldsymbol{u}\|^2]^\mathsf{T} \in \mathbb{R}^5, \tag{43a}$$

$$\boldsymbol{\theta} = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5]^\mathsf{T} \in \mathbb{R}^5, \tag{43b}$$

respectively, and the final cost $h(\boldsymbol{x}_{T+1}) = 100\big((q_1 - \frac{\pi}{2})^2 + q_2^2 + \dot{q}_1^2 + \dot{q}_2^2\big)$, as the robot arm aims to reach and stop at the pose of $\boldsymbol{q} = [\frac{\pi}{2}, 0]^\mathsf{T}$. The discrete-time horizon is $T = 50$.

We still 'simulate' directional corrections. Suppose that we know the true $\boldsymbol{\theta}^* = [1,1,1,1,1]^\mathsf{T}$. At each iteration k, the simulation generates a directional correction $\boldsymbol{a}_{t_k}$ using the sign of gradient of the true $J(\boldsymbol{\theta}^*)$:

$$\boldsymbol{a}_{t_k} = -\text{sign}\left(\nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*)\,[2t_k : 2t_k+1]\right) \in \mathbb{R}^2, \quad (44)$$

where $\nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*)\,[2t_k : 2t_k+1]$ are the entries at the locations from $2t_k$ to $2t_k+1$ in ∇J (because $\boldsymbol{u} \in \mathbb{R}^2$), and the correction time t_k is randomly drawn from (uniform distribution) time horizon $[0, T]$. `sign` is applied entry-wise.

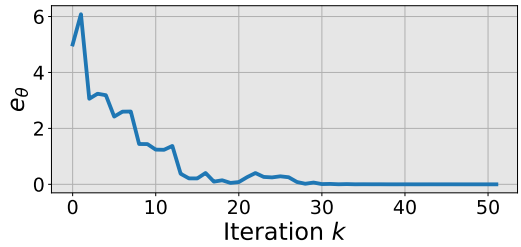


Fig. 8: Learning from simulated directional corrections for the robot arm. The plot is $e_{\boldsymbol{\theta}} = \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2$ versus iteration. The corrections are given in Fig. 9.

The initial weight search space is $\boldsymbol{\Omega}_0 = \{\boldsymbol{\theta} \mid 0 \le \boldsymbol{\theta}[i] \le 4,\ \ i = 1, 2, ..., 5\}$. We set the termination threshold $\epsilon = 10^{-1}$, and the maximum learning iteration is $K = 83$ by (35). We apply Algorithm 1 to learn the true $\boldsymbol{\theta}^*$. We define the weight error $e_{\boldsymbol{\theta}} = \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2$ and plot $e_{\boldsymbol{\theta}}$ versus iteration in Fig. 8. We also plot $\boldsymbol{a}_{t_k} = [a_{t_k,1}, a_{t_k,2}]^\mathsf{T}$ at each iteration k in Fig. 9, where the value labeled on each bar marks the correction time t_k. The results in Fig. 8 show that as the iteration increases, $\boldsymbol{\theta}_k$ converges to the true $\boldsymbol{\theta}^*$, which again confirms Theorem 1.

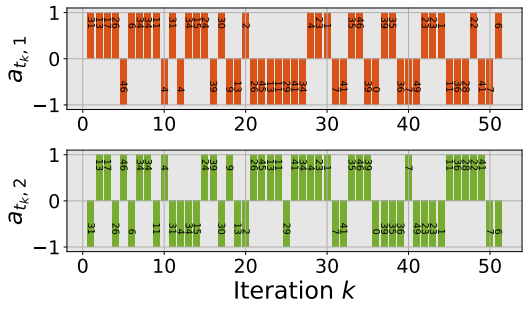


Fig. 9: Simulated directional correction $\boldsymbol{a}_{t_k} = [a_{t_k,1}, a_{t_k,2}]^\mathsf{T}$ at each iteration k for the robot arm. The number inside the bar is the correction time t_k randomly drawn from range $[0, 50]$.

C. Comparison with Magnitude-Correction Methods

We compare the proposed method with two related works [6], [7], both of which learn an objective function from magnitude corrections. The comparison is conducted on the inverted pendulum used in Section V-A. According to [7], given a magnitude correction, the trajectory deformation [8] is used to obtain a *human intended trajectory*. Specifically, suppose the robot trajectory is $\boldsymbol{\xi}_{\boldsymbol{\theta}_k} = \{\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}\}$. Given a correction $\bar{\boldsymbol{a}}_k$, the human intended trajectory, denoted as $\bar{\boldsymbol{\xi}}_{\boldsymbol{\theta}_k} = \{\bar{\boldsymbol{x}}_{0:T+1}^{\boldsymbol{\theta}_k}, \bar{\boldsymbol{u}}_{0:T}^{\boldsymbol{\theta}_k}\}$, is solved as

$$\bar{\boldsymbol{u}}_{0:T}^{\boldsymbol{\theta}_k} = \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k} + M^{-1}\bar{\boldsymbol{a}}_k, \quad (45)$$

where M is a matrix to propagate a single-time-step correction $\bar{\boldsymbol{a}}_k$ (5) along the rest of the robot trajectory [22], and $\bar{\boldsymbol{x}}_{0:T+1}^{\boldsymbol{\theta}_k}$ in $\bar{\boldsymbol{\xi}}_{\boldsymbol{\theta}_k}$ is obtained via rolling out the robot dynamics (1) given $\bar{\boldsymbol{u}}_{0:T}^{\boldsymbol{\theta}_k}$. The learning update used in both [6] and [7] is

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \eta\Big(\phi(\bar{\boldsymbol{\xi}}_{\boldsymbol{\theta}_k}) - \phi(\boldsymbol{\xi}_{\boldsymbol{\theta}_k})\Big), \quad (46)$$

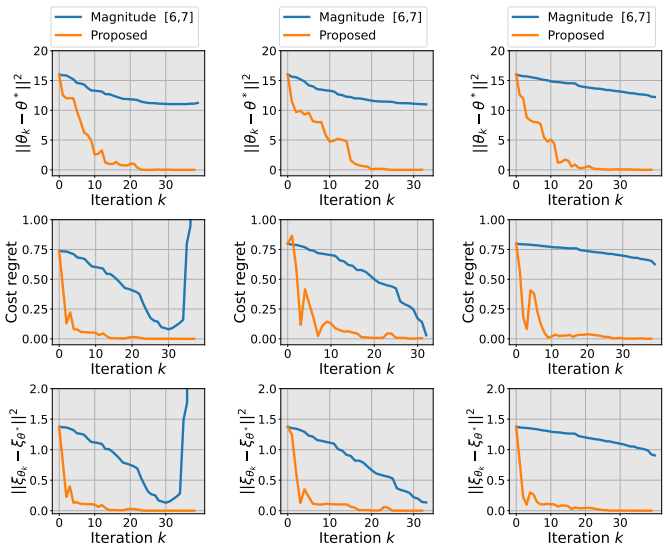
where $\phi(\bar{\boldsymbol{\xi}}_{\boldsymbol{\theta}_k})$ and $\phi(\boldsymbol{\xi}_{\boldsymbol{\theta}_k})$ are the feature vectors of the human intended trajectory $\bar{\boldsymbol{\xi}}_{\boldsymbol{\theta}_k}$ and the uncorrected robot trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$, respectively. Here, we set M as the finite differencing matrix [8] and $\eta = 0.0002$ (for best performance).

In comparison, we use the simulated corrections $\boldsymbol{a}_{t_k}$, which are generated from the true $\boldsymbol{\theta}^*$, as in (42). We set the magnitude $\|\boldsymbol{a}_{t_k}\|$ to three levels: $\|\boldsymbol{a}_{t_k}\| = 0.00125$, $\|\boldsymbol{a}_{t_k}\| = 0.001$, and $\|\boldsymbol{a}_{t_k}\| = 0.0008$, because we want to see how sensitive the magnitude-correction methods [6], [7] are to the magnitudes of corrections. Here, the correction time t_k for different magnitude levels is different random draws. To illustrate results, we measure the following three aspects for all methods.

- *Weight Error*, defined as $\|\boldsymbol{\theta}_k - \boldsymbol{\theta}^*\|^2$. This is to measure the error between the learned $\boldsymbol{\theta}_k$ and the true $\boldsymbol{\theta}^*$.
- *Cost Regret* [6], defined as $(\boldsymbol{\theta}^*)^\mathsf{T}\big(\phi(\boldsymbol{\xi}_{\boldsymbol{\theta}_k}) - \phi(\boldsymbol{\xi}_{\boldsymbol{\theta}^*})\big)$. This is to measure the misalignment of the costs between robot current trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ and the desired trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}^*}$, evaluated under the true cost function $J(\boldsymbol{\theta}^*) = (\boldsymbol{\theta}^*)^\mathsf{T}\phi(\boldsymbol{\xi})$.
- *Trajectory error*, defined as $\|\boldsymbol{\xi}_{\boldsymbol{\theta}_k} - \boldsymbol{\xi}_{\boldsymbol{\theta}^*}\|^2$, which measures the distance between the robot current trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ and the desired trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}^*}$.

Given the simulated corrections $\boldsymbol{a}_{t_k}$, we run Algorithm 1 and the method [6], [7] (i.e., updating the weight vector via (46)). We show the results in Fig. 10, where the results of our method are in orange and the results of the magnitude-correction method [6], [7] are in blue. Each column in Fig. 10 corresponds to one level of correction magnitudes (recall that we want to see how robust a method is against different magnitudes of corrections). In each magnitude level (each column), the first row shows the weight error versus iteration, the second row shows the cost regret versus iteration, and the third row shows the trajectory error versus iteration. Based on the results in Fig.10, we make the following comments.

For the magnitude-correction methods [6], [7] (in blue lines) in Fig. 10, comparing different columns, we note that larger magnitude leads to faster convergence. However, as shown by the left and middle panels in the first row, the magnitude-correction method converges to a local weight vector but not $\boldsymbol{\theta}^*$. This is not a surprise, because the magnitude-correction method can only guarantee the convergence of the cost regret, as shown by [6], not necessarily the convergence of the cost function itself. One conjecture to the above results could be that the local weight vector (different from the true $\boldsymbol{\theta}^*$) results in the same robot trajectory and the same optimal cost as $\boldsymbol{\theta}^*$ does. This conjecture has been evidenced in the second row and third row in Fig. 10, where the left and middle panels show that the magnitude-correction methods, although having



(a) $\|\boldsymbol{a}_{t_k}\| = 0.00125$ (b) $\|\boldsymbol{a}_{t_k}\| = 0.001$ (c) $\|\boldsymbol{a}_{t_k}\| = 0.0008$

Fig. 10: Comparison between the proposed method (in orange lines) and the magnitude-correction methods [6], [7] (in blue lines). Each column corresponds to a different level of correction magnitude, and correction time t_k in different magnitude levels are different random draws. For each magnitude level (each column), the first row shows the weight error versus iteration; the second row is the cost regret versus iteration; and the third row is the trajectory error versus iteration. Detailed analysis is given in the text.

learned a different weight vector, have both the cost regret and trajectory error converging to zero.

Also for the magnitude-correction methods [6], [7] (in blue lines), when $\|\boldsymbol{a}_{t_k}\| = 0.00125$ (first column), we observe that there is a sharp increase of the cost regret and trajectory error near convergence after 30 iterations. This could be because the magnitude-correction methods [6], [7] are not robust against the large magnitude of corrections near the desired trajectory. Specifically, after 30 iterations near convergence, a relatively larger magnitude, say $\|\boldsymbol{a}_{t_k}\| = 0.00125$, can be an over-correction and thus violating $J(\bar{\boldsymbol{u}}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*) < J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*)$. This explanation has been supported by the second column of Fig. 10, where with a smaller magnitude $\|\boldsymbol{a}_{t_k}\| = 0.001$, there is no such over-correction phenomenon.

Combining all three columns in Fig. 10, one can conclude that for the magnitude-correction methods [6], [7], large correction magnitude, such as $\|\boldsymbol{a}_{t_k}\| = 0.00125$, will have faster convergence, but it can lead to over-corrections, as shown in Fig. 10a, making the learning process unstable. Smaller correction magnitude, such as $\|\boldsymbol{a}_{t_k}\| = 0.0008$, can avoid the over-correction issue, but it leads to slow convergence, as shown in Fig. 10c. Notice that the magnitude change from $\|\boldsymbol{a}_{t_k}\| = 0.00125$ (unstable) in Fig. 10a to $\|\boldsymbol{a}_{t_k}\| = 0.0008$ (stable) in Fig. 10c is very small, which could suggest that the magnitude-correction methods are sensitive to the selection of correction magnitudes. Thus, it could be difficult in practice to provide a valid correction magnitude. We will further show and analyze this in our following user study in Section VI.

In contrast, Fig. 10 has shown that the proposed method consistently learns the true $\boldsymbol{\theta}^*$ and achieves a zero cost regret and zero trajectory error regardless of the correction magnitude levels. Also, compared to the magnitude-correction methods [6], [7], the proposed method requires less learning iterations (corrections) for convergence. Since the proposed method only leverages the direction of $\boldsymbol{a}_{t_k}$ regardless of $\|\boldsymbol{a}_{t_k}\|$, there is no over-correction issues near convergence. In practice, the choice of directional corrections is more flexible than choosing magnitude corrections, as analyzed in Fig. 3 in Section II. We will continue to show this advantage in the following user study in Section VI.

VI. User Study

To show the effectiveness of the proposed method for learning from real human directional corrections, we have developed two human-robot simulation games based on which a user study is conducted. One game is robot arm reaching (Fig. 11) and the other is 6-DoF quadrotor maneuvering (Fig. 12). In each game, a human participant observes the visualization of robot motion, while applying directional corrections through a keyboard. The goal of each game is to teach a robot to learn an objective function, such that it can successfully navigate through an environment without the knowledge about the environment obstacles. We have released the accompanying codes of those two games for the readers to try themselves: https://github.com/wanxinjin/Learning-from-Directional-Corrections.

In what follows next, Sections VI-A and VI-B describe the designs of the two games. Section VI-C provides the details of human participants and the procedure of the user study. The outcomes of the user study are presented in Section VI-D and detailed analysis is given in Section VI-E.

A. Robot Arm Reaching Game

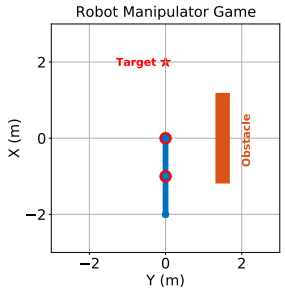


Fig. 11: The robot arm reaching game. A human participant teaches the robot arm to learn a cost function by applying directional corrections via a keyboard (see the instructions in Table I). The goal is to let the robot arm successfully move from the initial pose (the pose in the figure) to reach the target (the star) while avoiding the obstacle. Note that we have no knowledge about the location and size of the obstacle.

1) Robot Setup: The dynamics of a robot arm and its physical parameters follow Section V-B. The robot initial state is $\boldsymbol{x}_0 = [-\frac{\pi}{2}, 0, 0, 0]^\mathsf{T}$, as in Fig. 11. The parameterized cost function $J(\boldsymbol{\theta})$ in (2) is

$$\boldsymbol{\phi} = [q_1^2,\, q_1,\, q_2^2,\, q_2,\, \|\boldsymbol{u}\|^2]^\mathsf{T} \in \mathbb{R}^5, \tag{47a}$$

$$\boldsymbol{\theta} = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5]^\mathsf{T} \in \mathbb{R}^5. \tag{47b}$$

Here, $\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\phi}$ is a general second-order polynomial. It is worth noting that in practice if prior knowledge about good features is not available, general polynomial features are a good option. The final cost $h(\boldsymbol{x}_{T+1})$ in (2) is

$$h(\boldsymbol{x}_{T+1}) = 100\big((q_1 - \frac{\pi}{2})^2 + q_2^2 + \dot{q}_1^2 + \dot{q}_2^2\big), \tag{48}$$

as the robot arm aims to finally reach and stop at the target pose: $q_1^{\text{target}} = \frac{\pi}{2}$ and $q_2^{\text{target}} = 0$, marked in Fig. 11. The time horizon of the game is set as $T = 50$ (that is, $50\Delta = 10$s). Since we choose the polynomial features (47a), $\boldsymbol{\theta}$ will dictate how the robot reaches the target (i.e., the specific trajectory to the goal). By default, the initial search space $\boldsymbol{\Omega}_0$ is set

$$\boldsymbol{\Omega}_0 = \{\boldsymbol{\theta} \,|\, \theta_1, \theta_3 \in [0, 1], \theta_2, \theta_4 \in [-3, 3], \theta_5 \in [0, 0.5]\}. \tag{49}$$

Without human corrections, the robot with a random $\boldsymbol{\theta}_0$ will move and crash into the obstacle in Fig. 11. The goal of the game is to let a human participant teach the robot arm to learn a cost function, such that it can move from the initial condition and reach the target pose while avoiding the obstacle. Note that we have no knowledge about the obstacle.

TABLE I: Keyboard interface for the robot arm game.

Keys	Directional correction	Interpretation of correction
up	$\boldsymbol{a} = [1, 0]$	counter-close-wise torque to Joint 1
down	$\boldsymbol{a} = [-1, 0]$	close-wise torque to Joint 1
left	$\boldsymbol{a} = [0, 1]$	counter-close-wise torque to Joint 2
right	$\boldsymbol{a} = [0, -1]$	close-wise torque to Joint 2

* When implementing [6], [7], to allow magnitude corrections via the same interface, we additionally detect the key pressing duration δt and interpret the correction magnitude u to be proportional to δt with saturation $u_{\max}$; i.e., $u = \min\{\beta\delta t, u_{\max}\}$ with $\beta = 2$ and $u_{\max} = 1$. The magnitude correction using the above interface is $\boldsymbol{u} = u\boldsymbol{a}$. The correction time t_k is the beginning time of the key press.

2) Interface: In the robot arm game, we use a keyboard as the interface for a human participant to apply directional corrections. We customize (*up, down, left, right*) keys and associate them with corresponding directional corrections as listed in Table I. During the game, at each learning iteration, a human participant is allowed to press one or multiple keys from (*up, down, left, right*), and the interface is listening to which key(s) the human player hits and recording the timing of the keystroke(s). The recorded information is translated into the directional correction $\boldsymbol{a}_{t_k}$ as per Table I. For example, at iteration k, while the robot is executing the motion trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$, a human player hits the *up* and *left* keys simultaneously at the time step 10; then the corresponding correction information is translated into $\boldsymbol{a}_{t_k} = [1, 1]^\mathsf{T}$ with $t_k = 10$ according to Table I. We obtain averaged $\bar{\boldsymbol{a}}_k$ from $\boldsymbol{a}_{t_k}$ via (5) and (6b).

B. 6-DoF Quadrotor Maneuvering Game

1) Robot Setup: The dynamics of a quadrotor is

$$\dot{\boldsymbol{r}}_I = \dot{\boldsymbol{v}}_I, \tag{50a}$$

$$m\dot{\boldsymbol{v}}_I = m\boldsymbol{g}_I + \boldsymbol{f}_I, \tag{50b}$$

$$\dot{\boldsymbol{q}}_{B/I} = \frac{1}{2}\Omega(\boldsymbol{\omega}_B)\boldsymbol{q}_{B/I}, \tag{50c}$$

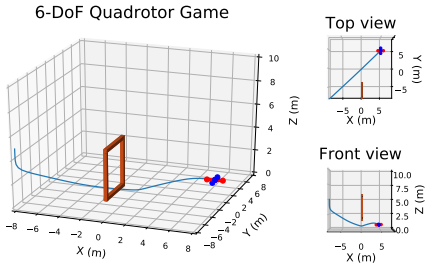


Fig. 12: The 6-DoF quadrotor game. A human participant teaches the quadrotor to learn a cost function by applying directional corrections through a keyboard (see the instruction in Table II). The goal is that the quadrotor can fly from the initial position (in the bottom left), go through a gate (colored in brown), and land on the target (in the upper right). Note that we have no knowledge about the gate.

$$J_B\dot{\boldsymbol{\omega}}_B = \boldsymbol{\tau}_B - \boldsymbol{\omega}_B \times J_B\boldsymbol{\omega}_B. \tag{50d}$$

Here, the subscripts $_B$ and $_I$ denote the quantities expressed in the quadrotor's body frame and world frame, respectively; m is the mass of the quadrotor; $\boldsymbol{r}_I \in \mathbb{R}^3$ and $\boldsymbol{v}_I \in \mathbb{R}^3$ are the position and velocity, respectively; $J_B \in \mathbb{R}^{3\times3}$ is the moment of inertia; $\boldsymbol{\omega}_B \in \mathbb{R}^3$ is the angular velocity; $\boldsymbol{q}_{B/I} \in \mathbb{R}^4$ is the unit quaternion [31] describing the attitude of the quadrotor's body frame with respect to the world frame; (50c) is the time derivative of the quaternion with $\Omega(\boldsymbol{\omega}_B)$ being the matrix form of $\boldsymbol{\omega}_B$ for quaternion multiplication [31]; $\boldsymbol{\tau}_B \in \mathbb{R}^3$ is the torque vector applied to the quadrotor; and $\boldsymbol{f}_I \in \mathbb{R}^3$ is the total force vector applied to the center of mass (COM). The total force magnitude $\|\boldsymbol{f}_I\| = f$ (along the z-axis of the quadrotor's body frame) and the torque $\boldsymbol{\tau}_B = [\tau_x, \tau_y, \tau_z]^\mathsf{T}$ are generated by four rotor thrusts $[T_1, T_2, T_3, T_4]^\mathsf{T}$ as follows:

$$\begin{bmatrix} f \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -l_\text{w}/2 & 0 & l_\text{w}/2 \\ -l_\text{w}/2 & 0 & l_\text{w}/2 & 0 \\ \kappa & -\kappa & \kappa & -\kappa \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix}, \tag{51}$$

with l_w denoting the wing length of the quadrotor and κ a fixed constant, here $\kappa = 0.1$. In dynamics (50), the gravity constant $\|\boldsymbol{g}_I\|$ is set as 10m/s^2 and the other physical parameters are units. We define the state vector of the quadrotor as

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{r}_I & \boldsymbol{v}_I & \boldsymbol{q}_{B/I} & \boldsymbol{\omega}_B \end{bmatrix} \in \mathbb{R}^{13}, \tag{52}$$

and the control input vector as

$$\boldsymbol{u} = \begin{bmatrix} T_1 & T_2 & T_3 & T_4 \end{bmatrix}^\mathsf{T} \in \mathbb{R}^4. \tag{53}$$

We discretize (50) by the Euler method with interval $\Delta = 0.1$s. To achieve $SE(3)$ maneuvering, we define the attitude error between the quadrotor attitude $\boldsymbol{q}$ and target $\boldsymbol{q}^{\text{target}}$ as [32]

$$e(\boldsymbol{q}, \boldsymbol{q}_{\text{target}}) = \frac{1}{2}\text{trace}(I - R^\mathsf{T}(\boldsymbol{q}^{\text{target}})R(\boldsymbol{q})), \tag{54}$$

where $R(\boldsymbol{q}) \in \mathbb{R}^{3\times3}$ is the rotation matrix corresponding to $\boldsymbol{q}$.

In the cost function (2), we set the final cost term as

$$h(\boldsymbol{x}_{T+1}) = \|\boldsymbol{r}_I - \boldsymbol{r}_I^{\text{target}}\|^2 + 10\|\boldsymbol{v}_I\|^2 +$$

$$100e(\boldsymbol{q}_{B/I}, \boldsymbol{q}_{B/I}^{\text{target}}) + 10\|\boldsymbol{w}_B\|^2, \qquad (55)$$

because the quadrotor aims to finally land on a target position $\boldsymbol{r}_I^{\text{target}}$ with a target attitude $\boldsymbol{q}_{B/I}^{\text{target}}$. Here, $\boldsymbol{r}_I = [r_x, r_y, r_z]^\mathsf{T}$ is the position of the quadrotor expressed in the world frame. We set the weight-feature cost term as

$$\boldsymbol{\phi} = \begin{bmatrix} r_x^2 & r_x & r_y^2 & r_y & r_z^2 & r_z & \|\boldsymbol{u}\|^2 \end{bmatrix}^\mathsf{T} \in \mathbb{R}^7, \qquad (56a)$$

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 & \theta_4 & \theta_5 & \theta_6 & \theta_7 \end{bmatrix}^\mathsf{T} \in \mathbb{R}^7. \qquad (56b)$$

Here, feature vector $\boldsymbol{\phi}$ consists of general polynomial features and $\boldsymbol{\theta}$ will determine the specific trajectory of the quadrotor to the target. By default, the initial weight search space is

$$\boldsymbol{\Omega}_0 = \{\boldsymbol{\theta} \,|\, \theta_1, \theta_3, \theta_5 \in [0, 1], \\ \theta_2, \theta_4, \theta_6 \in [-8, 8], \theta_7 \in [0, 0.5]\}. \qquad (57)$$

As in Fig. 12, the goal of this quadrotor game is to let a human participant teach the quadrotor to learn a cost function, such that the quadrotor can successfully fly from the initial position $\boldsymbol{r}_I(0) = [-8, -8, 5]^\mathsf{T}$ (in the bottom left), go through a gate (in brown color), and finally land on a target position $\boldsymbol{r}_I^{\text{target}} = [8, 8, 0]^\mathsf{T}$ (in the upper right) with the target attitude $\boldsymbol{q}_{B/I}^{\text{target}} = [1, 0, 0, 0]^\mathsf{T}$. The initial attitude of the quadrotor is $\boldsymbol{q}_{B/I}(0) = [1, 0, 0, 0]^\mathsf{T}$ and initial velocities zeros. The time horizon for this game is $T = 50$, that is, $T\Delta = 5$s. Note that we have no knowledge about the gate.

TABLE II: Keyboard interface for 6-DoF quadrotor game.

Keys	Directional correction	Interpretation of correction
'up'	$T_1{=}1, T_2{=}1,$ $T_3{=}1, T_4{=}1$	Upward force applied at COM
'down'	$T_1{=}-1, T_2{=}-1,$ $T_3{=}-1, T_4{=}-1$	Downward force applied at COM
'w'	$T_1{=}0, T_2{=}1$ $T_3{=}0, T_4{=}-1$	Negative torque along body-axis x
's'	$T_1{=}0, T_2{=}-1$ $T_3{=}0, T_4{=}1$	Positive torque along body-axis x
'a'	$T_1{=}1, T_2{=}0$ $T_3{=}-1, T_4{=}0$	Negative torque along body-axis y
'd'	$T_1{=}-1, T_2{=}0$ $T_3{=}1, T_4{=}0$	Positive torque along body-axis y

* When implementing [6], [7], to allow magnitude corrections via the same interface, we additionally detect key pressing duration δt and interpret the correction magnitude u to be $u = \min\{\beta\delta t, u_{\max}\}$ with $\beta{=}2$ and $u_{\max}{=}1$ in our implementation. Thus, the magnitude correction using the above interface is $\boldsymbol{u} = u[T_1, T_2, T_3, T_4]^\mathsf{T}$. The correction time t_k is the beginning time of the key press.

2) Interface: The keyboard interface for applying directional corrections is in Table II. Specifically, we customize the ('*up*', '*down*', '*w*', '*s*', '*a*', '*d*') keys and map them to specific directional correction signals. During a learning iteration, a human participant is allowed to press one or multiple keys in Table II. The interface listens to the keystrokes and translates the keystrokes into the directional corrections based on Table II. The time step at which a key is hit is the correction time t_k. For example, if a human participant presses *'s'* key at time step 5; then, the directional correction will be $\boldsymbol{a}_{t_k} = [0, -1, 0, 1]^\mathsf{T}$ with $t_k = 5$. Based on (5) and (6b), we obtain $\bar{\boldsymbol{a}}_k$ from $\boldsymbol{a}_{t_k}$.

C. Participants and Procedure

A total of 17 volunteers from Purdue College of Engineering have participated in our user study. Among these 17 participants, 1 was female and 16 male, with ages from 20 to 37 years old (25.812 ± 3.936). 5 of those 17 participants had no robotics or related background, and all participants were novices to the two games. This user study had been reviewed and approved by the Institutional Review Board (IRB) of Purdue University, and all participants had signed the consent forms.

Each participant was instructed to play the above two games. In each game, a participant played 5 successive rounds with the proposed method and 5 successive rounds with the magnitude-correction method [6], [7]. *A game round* is defined as a complete run of a learning method until the success or failure of the round. A failure of a game round is identified if the robot fails to accomplish the task after the total number of human corrections has exceeded the maximum correction count. In our user study, the maximum correction count is 20 for the robot arm game and 15 for the quadrotor game. We set the maximum correction count because the participants are expected to try their best to teach the robot with as few corrections as possible.

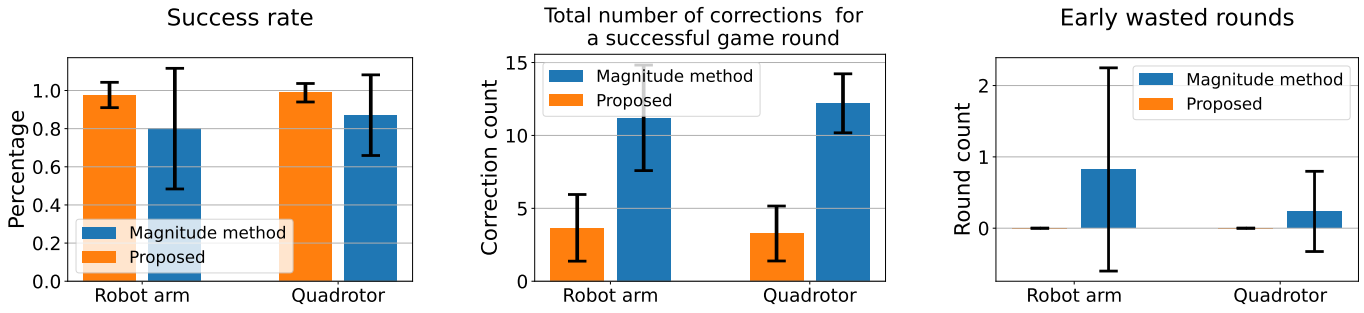
We used a within-subjects design: the order of the two games (i.e., which game goes first) and learning methods used (i.e., which method is used first) were counterbalanced across all participants in our user study. Also, the participants were not informed which methods were being used for the current game. Before starting a game, each participant was instructed how to play the game and was given one hands-on game round to get familiar with the interface before the experiment recording starts. After the games, each participant was asked to finish a post-game survey about her/his opinions of the game experience (which will be reported in Section VI-E4).

D. Outcome Measurements

For each game (robot arm or quadrotor) with each learning method (the proposed method or magnitude-correction method [6], [7]), we measure the following outcomes for one participant in his/her 5-successive game rounds.

- *Success rate*. This is the ratio of successful game rounds over all rounds attempted (which is 5 here). This outcome indicates the **efficacy** of a learning method.
- *Total number of corrections for a successful game round.* This measures how many human corrections are needed for a successful game round. This measure can show the **efficiency/effortlessness** of a learning method—fewer corrections mean less human effort in teaching a robot.
- *Early wasted rounds.* This is to measure how many early rounds of a game are wasted (failed) before a participant begins to constantly play successful game rounds. This can show the **accessibility** of a learning method—fewer early wasted rounds indicate a more accessible experience for a participant to successfully teach a robot.

The outcomes of all participants are organized according to the method type (the proposed method and magnitude-correction method [6], [7]) and the game type (robot-arm and quadrotor).



(a) Success rate (b) Total correction count in a success game (c) Early wasted rounds

Fig. 13: The outcomes of all 17 participants for playing different robot games (robot-arm or quadrotor) using different learning methods (the proposed method and magnitude-correction method [6], [7]). (a) shows the success rate, i.e., the ratio of successful game rounds over total 5 rounds attempted; (b) shows the total number of human corrections needed for one successful game round; and (c) shows the number of the early wasted (failed) game rounds before a participant constantly play successfully. The bars denote the mean values over all participants, and the top line segments are the standard deviations. Analysis based on statistical tests (with significance level $\alpha = 0.05$) is in Section VI-E. One illustrative round for the robot arm game from one participant is in Fig. 14 and Table III, and one illustrative quadrotor game round from a participant is in Fig. 15 and Table IV.

We report all outcomes in Fig. 13 and provide a detailed analysis of the results based on statistical tests.

E. Results and Analysis

The statistics of three outcomes over all 17 participants are shown in Fig. 13. Each outcome is presented with respect to each learning method (the proposed method or magnitude-correction method [6], [7]) on each game (robot arm or quadrotor). All three outcomes here are averaged over all participants with the error bar denoting the standard deviation over all participants. We have the following analysis of the results based on statistical tests with significance level $\alpha = 0.05$,

1) Success Rate: In Fig. 13a, for the robot arm game, the proposed method obtains a success rate of $97.65\% \pm 6.64\%$ compared to the magnitude-correction method [6], [7]'s $80.00\% \pm 31.62\%$. The proposed method yields a significantly higher success rate than the magnitude-correction method with t-score $t = 2.25$ and p-value $p = 0.031 < \alpha$. For the quadrotor game in Fig. 13a, the proposed method attains $94\% \pm 12.8\%$ versus the magnitude-correction method's $82\% \pm 20.9\%$. It also shows a significantly higher success rate than the magnitude-correction method [6], [7], with t-score $t = 2.24$ and p-value $p = 0.032 \leq \alpha$.

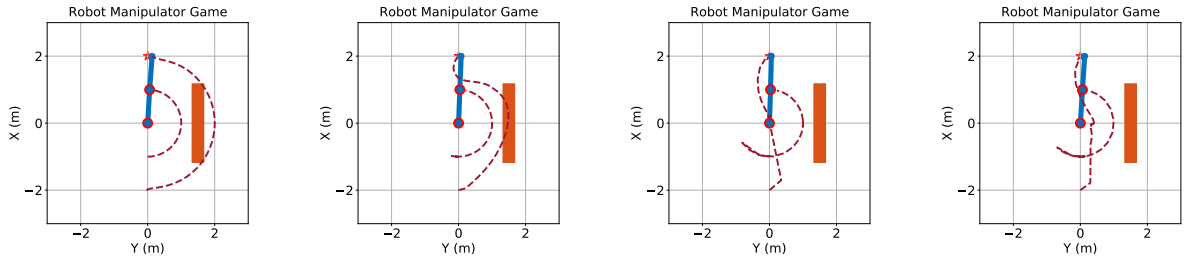
We provide the following comments for the above results. Recall that the magnitude-correction method [6], [7] assumes $J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k} + \bar{\boldsymbol{a}}_k, \boldsymbol{\theta}^*) < J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*)$, while the our method assumes $\langle -\nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*), \bar{\boldsymbol{a}}_k \rangle > 0$, regardless of magnitude $\|\bar{\boldsymbol{a}}_k\|$. As shown in Fig. 3, *the allowable region of directional corrections is much larger than that of magnitude corrections*. Since all participants are novices to both games, giving valid magnitude correction could be difficult for them. By contrast, novice participants are more likely to give valid directional corrections. This leads to higher success rate and lower variance for the proposed method than the magnitude-correction method.

The difficulty of giving valid magnitude corrections can also be seen from the number of early wasted/failed rounds in Fig. 13c. For the magnitude method, before a participant masters a game, 0.82 ± 1.42 early game rounds are wasted/failed for the robot arm game, and 0.24 ± 0.56 early rounds are failed for the quadrotor game. In contrast, the proposed method has 0 ± 0 early wasted rounds in both games. This means that a novice participant needs more practice to master the skill of giving valid magnitude corrections.

2) Total Number of Corrections for Successful Game: In Fig. 13b, for a successful robot arm game, the proposed method needs a total of 3.66 ± 2.29 corrections, while the magnitude-correction method [6], [7] requires 11.21 ± 3.62 corrections. Thus, the proposed method requires significantly fewer corrections than the magnitude-correction method, with t-score $t = 7.27$ and p-value $p = 2.94 \times 10^{-8} \leq \alpha$. In the quadrotor game, the proposed method takes a total of 3.27 ± 1.88 corrections for a successful round, while the magnitude-correction method needs 12.20 ± 2.02 corrections. This again shows that the proposed method is significantly more efficient (fewer corrections) than the magnitude-correction method, with t-score $t = 13.32$ and p-value $p = 1.35 \times 10^{-14} < \alpha$.

We have the following interpretations for the above results. Recall that the assumption for a valid magnitude correction $\bar{\boldsymbol{a}}_k$ is $J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k} + \bar{\boldsymbol{a}}_k, \boldsymbol{\theta}^*) < J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*)$. Since all participants are novice to the games, the magnitude corrections made by a participant were not always valid in a game round; in other words, some magnitude corrections can be over-correction. Because of those invalid corrections, the magnitude-correction method takes longer (more corrections) to succeed. In contrast, the proposed method only leverages directional corrections, which are more likely to satisfy $\langle -\nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*), \bar{\boldsymbol{a}}_k \rangle > 0$; it requires fewer human corrections (effort).

Since the total number of corrections for a successful game indicates how much effort a participant needs to successfully teach a robot. Thus, the above results in Fig. 13b show that the proposed method is significantly more effortless than the magnitude-correction method [6], [7].

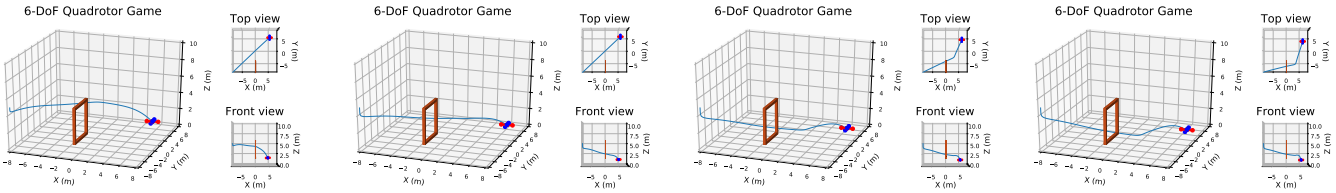


(a) Iteration $k = 1$ (b) Iteration $k = 2$ (c) Iteration $k = 3$ (d) Iteration $k = 4$

Fig. 14: One illustrative round for the robot arm game from one participant. At each iteration k, the robot weight vector guess $\boldsymbol{\theta}_k$ and the participant directional correction $\boldsymbol{a}_{t_k}$ are reported in Table III.

TABLE III: An illustrative round for the robot arm game from one participant.

Iteration k	Current weight vector guess $\boldsymbol{\theta}_k$	A participant's directional correction $\boldsymbol{a}_{t_k}$ and correction time t_k
$k = 1$	$\boldsymbol{\theta}_k = [0.50,\ 0.00,\ 0.50, -0.00,\ 0.25]^\mathsf{T}$	$\boldsymbol{a}_{t_k} = [0, 1]$ (i.e., *left* key pressed) and $t_k = 11$
$k = 2$	$\boldsymbol{\theta}_k = [0.50,\ 0.00,\ 0.50, -1.50,\ 0.25]^\mathsf{T}$	$\boldsymbol{a}_{t_k} = [0, 1]$ (i.e., *left* key pressed) and $t_k = 16$
$k = 3$	$\boldsymbol{\theta}_k = [0.50,\ 0.00,\ 0.34, -2.03,\ 0.25]^\mathsf{T}$	$\boldsymbol{a}_{t_k} = [-1, 0]$ (i.e., *down* key pressed) and $t_k = 34$
$k = 4$	$\boldsymbol{\theta}_k = [0.50,\ 1.48,\ 0.36, -2.00,\ 0.25]^\mathsf{T}$	*Game success!*



(a) Iteration $k = 2$ (b) Iteration $k = 3$ (c) Iteration $k = 4$ (d) Iteration $k = 5$

Fig. 15: One illustrative round for the quadrotor game from one participant. At each iteration k, the quadrotor weight vector guess $\boldsymbol{\theta}_k$ and the participant directional correction $\boldsymbol{a}_{t_k}$ are in Table IV. Iteration $k = 1$ is shown in Fig. 12.

TABLE IV: An illustrative round for the quadrotor game from one participant (Iteration $k = 1$ is shown in Fig. 12).

Iteration k	Current weight vector guess $\boldsymbol{\theta}_k$	A participant's directional corrections $\boldsymbol{a}_{t_k}$ and correction time t_k
$k = 1$	$\boldsymbol{\theta}_k = [0.50,\ 0.00,\ 0.50, -0.00,\ 0.50,\ -0.00,\ 0.25]^\mathsf{T}$	$\boldsymbol{a}_{t_k} = [1, 1, 1, 1]$ (i.e., *up* key pressed) and $t_k = 8$ $\boldsymbol{a}_{t_k} = [1, 1, 1, 1]$ (i.e., *up* key pressed) and $t_k = 20$
$k = 2$	$\boldsymbol{\theta}_k = [0.50,\ -0.00,\ 0.50, -0.00,\ 0.50,\ -3.99,\ 0.25]^\mathsf{T}$	$\boldsymbol{a}_{t_k} = [-1, -1, -1, -1]$ (i.e., *down* key pressed) and $t_k = 14$
$k = 3$	$\boldsymbol{\theta}_k = [0.50,\ -1.70,\ 0.50, -1.70,\ 0.52,\ -1.89,\ 0.25]^\mathsf{T}$	$\boldsymbol{a}_{t_k} = [0, -1, 0, 1]$ (i.e., *'s'* key pressed) and $t_k = 13$
$k = 4$	$\boldsymbol{\theta}_k = [0.50,\ -2.76,\ 0.50, -2.45,\ 0.60,\ -2.22,\ 0.25]^\mathsf{T}$	$\boldsymbol{a}_{t_k} = [0, -1, 0, 1]$ (i.e., *'s'* key pressed) and $t_k = 19$
$k = 5$	$\boldsymbol{\theta}_k = [0.50,\ -3.11,\ 0.50,\ 4.89,\ 0.65\ -2.67,\ 0.25]^\mathsf{T}$	*Game success!*

3) Early Wasted Rounds: In Fig. 13c, in the robot arm game (5 rounds in total), the early wasted (failed) rounds is 0 ± 0 for the proposed method and 0.82 ± 1.42 for the magnitude-correction method [6], [7]; thus, the proposed method requires significantly fewer rounds of practice (thus is more accessible) than the magnitude-correction method, with t-score $t = 2.38$ and p-value $p = 0.02 < \alpha$. For the quadrotor game (5 rounds in total), a participant wasted 0 ± 0 early rounds using the proposed method, while 0.24 ± 0.56 early rounds using the magnitude-correction method. The early wasted rounds of the proposed method are not significantly lower than that of the magnitude-correction method, with t-score $t = 1.73$ and p-value $p = 0.09 > \alpha$.

We have the following explanations for the above results. For the magnitude-correction method, since not all magnitude corrections are valid, a participant needs more early trials to realize the effects of correction magnitude on the robot motion and how to apply valid magnitude. Instead, for the proposed method, a participant only gives directional commands, which can be more intuitive, as shown in Fig. 14 and Fig. 15. Fig. 13c shows that the proposed method is potentially more accessible than the magnitude-correction method [6], [7]

4) Other Observations and Subjective Survey: We also observe that the number of the directional corrections in the user study is generally smaller than that in the simulation cases in Section V. This is because the convergence of the robot motion trajectory (to the desired motion in the human's perspective) is empirically faster than the convergence of the cost function

weight vector (to the true $\boldsymbol{\theta}^*$). This has been shown in Fig. 10, where the convergence of $\|\boldsymbol{\theta}_k - \boldsymbol{\theta}^*\|^2$ requires around 20 iterations, while the convergence of $\|\boldsymbol{\xi}_{\boldsymbol{\theta}_k} - \boldsymbol{\xi}_{\boldsymbol{\theta}^*}\|^2$ requires only 10 iterations. Since a game is deemed successful as long as the robot motion can successfully navigate through the environment, thus a participant usually takes fewer corrections. We also find that there is no unique solution for the choice of directional corrections and that different participants applied different corrections to manage the games.

We have also performed a post-game survey about how participants perceived the two learning methods. 10 out of 17 participants said that they felt *the robot with the proposed method is much 'smarter' than the robot with the magnitude-correction method [6], [7].* Those opinions are also consistent with the results in Fig. 13, as we have analyzed above.

5) Summary: In summary of all statistical analysis above, we conclude that compared to the state-of-the-art magnitude-correction method [6], [7], the proposed method is

- *significantly more effective* ($p < 0.033$) — it attains higher success rate for teaching robots,
- *significantly more efficient* ($p < 10^{-7}$) — fewer human corrections are needed for successful robot learning,
- *potentially more accessible* ($p \leq 0.09$) for human users — a novice human user takes fewer trials before constantly providing successful corrections to a robot.

The post-game survey reports that a majority of the user-study participants indicated that a robot with the proposed method is much 'smarter' than with the magnitude-correction method.

VII. Real-world Experiment

In this section, we test the proposed method in a real-world experiment using a Parrot Mambo quadrotor.

(a) Main camera view (b) Back camera view

Fig. 16: The setup of the real-world experiment. A human user teaches the quadrotor to fly from an initial position (red circle), go through the gate (with yellow label), and finally, land on a target position (green circle). Without human corrections, the quadrotor is initialized with a random cost function and fails to accomplish the task. Note that we have no knowledge about the gate.

A. Experiment Setup and Procedure

Our real-world experiment is a Parrot Mambo quadrotor task shown in Fig. 16. The goal of the experiment is to let a human user teach the quadrotor to accomplish the following task: starting from the initial position (red circle), flying through the gate (labeled in yellow), and landing on the target position (green circle). Without human correction, the quadrotor with a random initial cost function fails to accomplish the task. The procedure of learning from human directional corrections is as follows:

- First, perform the quadrotor motion planning in a remote computer by solving a trajectory optimization with the current cost function;
- Second, the quadrotor executes the plan (tracking control), at the same time a human user watches the quadrotor's motion and applies directional corrections via the keyboard of the remote computer.
- Third, update the quadrotor's cost function in the remote computer according to Algorithm 1; then repeat the above steps until the success of the task.

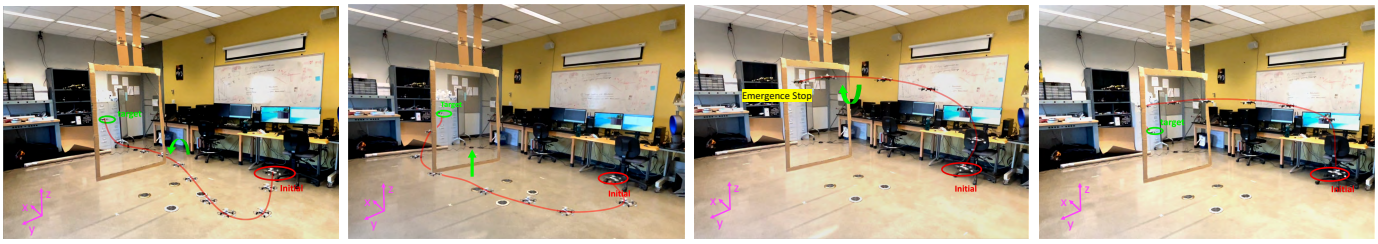
In the above procedure, the interface for a human user to apply directional corrections is still a keyboard, following Table II. The communication frequency between the quadrotor and the remote computer is around 10Hz. Note that because the cost function in some learning iterations may lead the quadrotor to collide with the gate during execution, we have included a collision stop mechanism: if the quadrotor detects that it is too close to the gate, an emergence stop is triggered. In the iteration where an emergence stop is triggered, human corrections, if applied before an emergence stop, can still count for updating the cost function for the next iteration. We will discuss the damage protection of robot learning in details in Appendix B-C.

We have invited a human user (not the co-authors themselves), who is a novice to our work, to perform the above experiment. The human user first had a successful warm-up training in our previous quadrotor game in order to get familiar with the interface (Table II), and then successfully conducted this real-world quadrotor experiment in just one run.

B. Results and Analysis

The results in all learning iterations are in Fig. 17. At each iteration, the quadrotor's trajectory is highlighted in the red lines, and the human directional corrections are labeled in the green arrows. The initial and target positions are also marked. We provide the following analysis for the results in Fig. 17.

In the first iteration in Fig. 17a, since the quadrotor was initialized with a random cost function, the planned and executed trajectory (labeled in the red line) are off the gate—the quadrotor missed the gate and flew from its right side. Thus, when observing the quadrotor's motion, the human user applied a directional toque to correct the quadrotor, which is in the direction of the negative x-axis, as shown by the green arrow in Fig. 17a. After receiving the correction, the quadrotor updated its cost function and planned a new trajectory for the next iteration in Fig. 17b. In the second iteration in Fig. 17b, while the quadrotor was executing the new motion, the human user observed that its motion was too below the gate and thus applied a directional force in the positive z-axis. Using this directional correction, the quadrotor updated its cost function for the third iteration in Fig. 17c.



(a) Iteration $k = 1$ (b) Iteration $k = 2$ (c) Iteration $k = 3$ (d) Iteration $k = 4$

Fig. 17: Results of a real-world quadrotor learning from human directional corrections. Here, the quadrotor trajectory is marked in red lines. The initial position (red circle) and target position (green circle) are also marked. In (a), during the quadrotor flying, the human user applied a torque correction in the opposite of the x-axis, shown by the green arrow. In (b), during the quadrotor flying, the human user applied a upward thrust correction (in green arrow) in z-axis. In (c), the quadrotor detected a potential collision with the left pillar of the gate and thus triggered an emergence stop. But before the emergence stop, the human user had already applied a positive x-axis (in green arrow) torque to correct the quadrotor. In (d), the quadrotor successfully learned a cost function to fly from the initial position, go through the gate, and finally reach the target position.

In the third iteration in Fig. 17c, while executing the planned motion, the quadrotor detected a potential collision with the left pillar of the gate and thus immediately triggered an emergence stop. However, because humans are able to predict the potential collision, the human user had already applied a directional torque in the positive x-axis (the green arrow) *before the emergence stop*. Thus, although the quadrotor had stopped in emergence, the directional correction still counted in the update of the cost function. In the fourth iteration in Fig. 17d, the quadrotor successfully flew through the gate and reached the target position.

Thus, it took only three human directional corrections for the quadrotor to learn a cost function to accomplish the task in an unknown environment. Based on the above results and analysis, we can conclude that the proposed method is effective and efficient for learning an objective function for desired robot motion from human directional corrections.

VIII. Conclusion

This paper has proposed a new method to enable a robot to learn an objective function from human directional corrections. A human directional correction can be any input change to the robot as long as it is in a direction that improves the robot's current motion relative to an implicit objective function. The proposed learning method only uses the direction of a correction to update the estimate of the objective function. The learning process is based on the cutting plane method and has straightforward geometric interpretations. We have established the theoretical results to show the convergence of the estimate of the objective function towards the true one induced by all human corrections. The proposed approach has been validated by numerical examples, a user study on two human-robot games, and a real-world quadrotor experiment. The results confirm the convergence and efficacy of the proposed method and show that the proposed method is significantly more effective (higher success rate), efficient/effortless (less correction needed), and potentially more accessible (fewer early wasted trials) than the state-of-the-art human-robot learning methods.

Acknowledgments

We thank Tianyu Zhou for his effort in helping organize and collect the data in the user study.

Appendix A
Proof of Lemma 2

First, we prove (25). From Step 2 in the main algorithm, we know that the robot's current trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k} = \{\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}\}$ is a result of minimizing the cost function $J(\boldsymbol{\theta}_k)$ subject to dynamics constraint in (1). This means that $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ must satisfy the optimality condition (i.e., first order condition). Following a similar derivation from (12) to (19) in the proof of Lemma 1, we can obtain the optimality condition

$$\mathbf{0} = \nabla J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}_k) = \boldsymbol{H}_1(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})\boldsymbol{\theta}_k + \boldsymbol{H}_2(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})\nabla h(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{T+1}). \quad (58)$$

It is worth mentioning that the above optimality condition (58) is also derived in [5]. As a result,

$$0 = \left\langle \nabla J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}_k), \bar{\boldsymbol{a}}_k \right\rangle \quad (59)$$

$$= \left\langle \boldsymbol{H}_1(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})\boldsymbol{\theta}_k, \bar{\boldsymbol{a}}_k \right\rangle + \left\langle \boldsymbol{H}_2(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})\nabla h(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{T+1}), \bar{\boldsymbol{a}}_k \right\rangle \quad (60)$$

$$= \langle \boldsymbol{h}_k, \boldsymbol{\theta}_k \rangle + b_k, \quad (61)$$

where the third equality is due to the definition of hyperplane in (8). This completes the proof of (25).

Next, we prove (26) by induction. In the main algorithm, we know $\boldsymbol{\theta}^* \in \boldsymbol{\Omega}_0$ for $k = 0$. Assume that $\boldsymbol{\theta}^* \in \boldsymbol{\Omega}_{k-1}$ holds at the $(k-1)$-th iteration. By Step 3 in the main algorithm, we have

$$\boldsymbol{\Omega}_k = \boldsymbol{\Omega}_{k-1} \cap \{\boldsymbol{\theta} \in \mathbb{R}^r \mid \langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k < 0\}. \quad (62)$$

In order to prove $\boldsymbol{\theta}^* \in \boldsymbol{\Omega}_k$ we only need to show that

$$\langle \boldsymbol{h}_k, \boldsymbol{\theta}^* \rangle + b_k < 0, \quad (63)$$

which is true according to (7) in Lemma 1. Thus, $\boldsymbol{\theta}^* \in \boldsymbol{\Omega}_k$ also holds at the kth iteration. Thus, we conclude that (26) holds. This completes the proof of Lemma 2. □

APPENDIX B
DISCUSSION

A. Learning Non-convex Cost Functions

In our problem formulation and method development, we have not imposed any restriction on the convexity of the cost function (2). However, the proposed method can sufficiently handle both convex and non-convex cost functions. But for non-convex cost functions, we have the following comments.

For a general non-convex cost function (2), the robot desired motion can be a *local* minimum of this non-convex cost function. As long as all human corrections aim to drive the robot towards the same local minimum (i.e., the same desired motion), all theories and methods developed in the paper still apply. In the otherwise case, human corrections may not have a consistent goal. For example, in one iteration the human intends to drive the robot towards one local minimum, while in the other iteration, the human aims to drive the robot to a *different* local minimum—a different desired motion. In those inconsistent cases, the assumption (4) would be violated, thus potentially leading to the failure of the proposed method.

In sum, if the cost function (2) is non-convex, the desired robot motion $\boldsymbol{\xi}_{\boldsymbol{\theta}^*}$ can be a local minimum of the non-convex cost function $J(\boldsymbol{\theta}^*)$. The proposed method in this paper still applies if all human corrections aim to drive the robot towards the same local minimum $\boldsymbol{\xi}_{\boldsymbol{\theta}^*}$.

B. On-the-fly Implementation of the Proposed Method

In the previous experiments, each learning iteration requires a robot to start over the task. However, this is not necessary, and one can readily implement the proposed method in an on-the-fly manner. An on-the-fly implementation is given in Algorithm 2, where the changes compared to Algorithm 1 are highlighted in red. Specifically, in Algorithm 2, after the robot receives a human directional correction $\boldsymbol{a}_{t_k}$ at time step t_k, instead of starting over the task (i.e., returning to the very early initial condition $\boldsymbol{x}_0$), we can update the robot cost function immediately and plan the motion by starting from the robot *current* state $\boldsymbol{x}_{t_k}$ (on which the correction $\boldsymbol{a}_{t_k}$ is made). Thus, the robot will continue to execute the task from the latest state without starting over. For this on-the-fly implementation, all theories and other properties of our method remain unchanged.

C. Damage Prevention During Learning

Since an intermediate cost function during learning iterations can lead a robot to collision or damage, we briefly discuss how to avoid such scenarios.

One effective way to prevent damage and collision during robot learning is to add emergence stop mechanisms in the robot's lower-level control, as we have adopted for our real-world quadrotor experiment in Section VII. Fortunately, as long as a human user applies directional corrections before the emergence stop, those corrections still count for the update of the cost function. Based on the previous user study and real-world experiment, we observe that humans usually have a good ability to predict potential collisions and are able to make preemptive corrections *before the robot triggers emergence stop*. Thus, it is a usual case that a robot has already received human directional corrections before triggering an emergence stop, and hence, the update of the cost function continues. This has been shown in Fig. 17c in our real-world experiment in Section VII.

Algorithm 2: Learning from directional corrections (an on-the-fly implementation version)

Input: Specify a termination threshold ϵ and use it to compute the maximum iteration K by (35).
Initialization: Initial weight search space $\boldsymbol{\Omega}_0$ in (21), and initial robot state $\boldsymbol{x}_{t_0}$ with $t_0 = 0$
for $k = 1, 2, \cdots, K$ **do**
- Choose a weight vector guess $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$ by Lemma 3;
- Starting from the current robot state $\boldsymbol{x}_{t_{k-1}}$, plan a new robot motion $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ by solving a trajectory optimization with the cost function $J(\boldsymbol{\theta}_k)$ and dynamics (1);
- Starting from $\boldsymbol{x}_{t_{k-1}}$, the robot executes the planned motion trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ while receving human directional corrections $\bar{\boldsymbol{a}}_{t_k}$;
- Compute the matrices $\boldsymbol{H}_1(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})$ and $\boldsymbol{H}_2(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})$, and then compute the hyperplane and half space $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k < 0$ by (7)-(8);
- Update the weight search space by $\boldsymbol{\Omega}_k = \boldsymbol{\Omega}_{k-1} \cap \{\boldsymbol{\theta} \in \mathbb{R}^r \mid \langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k < 0\}$ by (22);

end
Output: $\boldsymbol{\theta}_K$.

D. Choice of $\boldsymbol{\Omega}_0$ and ϵ in Algorithm 1

The initial weight search space $\boldsymbol{\Omega}_0$ in (21) should include any possible true $\boldsymbol{\theta}^*$, i.e., $\boldsymbol{\theta}^* \in \boldsymbol{\Omega}_0$. Although this is hard to verify as $\boldsymbol{\theta}^*$ is usually unknown in practice, a good practice is to choose as large $\boldsymbol{\Omega}_0$ as possible (note that $\boldsymbol{\Omega}_0$ also needs to ensure the existence of solution $\boldsymbol{\xi}_{\boldsymbol{\theta}}$; this is the reason why in our experiments, the range of weights of the second-order terms in a polynomial is always positive). Alternatively, one can also use a trial-and-error procedure to set the initial $\boldsymbol{\Omega}_0$: first, try a small $\boldsymbol{\Omega}_0$; under this small $\boldsymbol{\Omega}_0$ if the final (converged) robot motion is not desired, increase the size of $\boldsymbol{\Omega}_0$ and repeat this process until satisfied. Our previous experiment experience has shown that choosing a good $\boldsymbol{\Omega}_0$ is not difficult. This is because even a small size of $\boldsymbol{\Omega}_0$ has enough expressiveness power to represent a good variety of robot motions. This empiricism is consistent with the recent results of learning implicit models [33]–[35] in the machine learning community, where it shows that simple objective or energy functions can have enough representation power.

In Algorithm 1, ϵ determines the accuracy of weight vector convergence, as stated in Theorem 1. The choice of ϵ depends on specific accuracy requirements, and a large ϵ will terminate the algorithm early. In fact, it is always easy to set ϵ using a reasonably small value, because our previous experiment, such as in Fig. 10, has shown that the convergence of the robot motion trajectory is much faster than the convergence of the objective function itself. This means that it is a usual case that one observes a good convergence of robot motion trajectory before the convergence of cost function reaches

termination condition. This empiricism has also been reported in the literature on inverse optimal control such as [4].

E. Free from Noisy Robot Execution

Finally, we would point out that in implementation, the noise in robot states and controls cannot directly enter into the proposed algorithm. Specifically, as stated in Algorithm 1 and our real-world experiment, the proposed algorithm decouples motion planning (i.e., plan $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ by solving a trajectory optimization) and the robot execution of $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$. The noise of the robot states and inputs can only enter into the robot execution stage, not the motion planning stage. While a human observes the robot's noisy execution of $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ and gives directional corrections $\bar{\boldsymbol{a}}_k$, the proposed algorithm updates the cost function using the *noise-free* $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ *taken from the motion planner* instead of from the robot's execution. Thus, during the entire learning process, the noise in the robot's actual execution will never enter into the algorithm, thus will not influence the learning results.

APPENDIX C
OTHER CHOICES OF $\boldsymbol{\theta}_k$

For the weight search space $\boldsymbol{\Omega}_{k-1} \subset \mathbb{R}^r$, we choose $\boldsymbol{\theta}_k$ as the center of Maximum Volume Ellipsoid (MVE) inscribe $\boldsymbol{\Omega}_{k-1}$. Other choices for $\boldsymbol{\theta}_k$ could be the center of gravity [36], the Chebyshev center [37], the analytic center [38], etc.

1) Center of Gravity: The center of gravity for a polytope $\boldsymbol{\Omega}$ is defined as

$$\boldsymbol{\theta}_{\text{cg}} = \frac{\int_{\boldsymbol{\Omega}} \boldsymbol{\theta} d\boldsymbol{\theta}}{\int_{\boldsymbol{\Omega}} d\boldsymbol{\theta}}. \tag{64}$$

Following [39], the volume reduction rate using the center of gravity is

$$\frac{\mathbf{Vol}(\boldsymbol{\Omega}_{k+1})}{\mathbf{Vol}(\boldsymbol{\Omega}_k)} \leq 1 - \frac{1}{e} \approx 0.63, \tag{65}$$

which may lead to faster convergence than the rate $(1 - 1/r)$ using the center of MVE. However, for a polytope described by a set of linear inequalities, it is more expensive to compute the center of gravity in (64) than to compute the center of MVE [27].

2) Chebyshev Center: The Chebyshev center is defined as the center of the largest Euclidean ball that lies inside the polytope $\boldsymbol{\Omega}$. Chebyshev center for a polytope can be efficiently computed by solving a linear program [28]. But Chebyshev center is not affinely invariant to the transformations of coordinates [27]. Thus, a linear mapping of features may lead to an inconsistent weight vector estimation.

3) Analytic Center: Given a polytope $\boldsymbol{\Omega} = \{\boldsymbol{\theta} \mid \langle \boldsymbol{h}_i, \boldsymbol{\theta} \rangle + b_i < 0,\ i = 1, \cdots, m\}$, the analytic center is defined as

$$\boldsymbol{\theta}_{\text{ac}} = \min_{\boldsymbol{\theta}} - \sum\nolimits_{i=1}^{m} \log(b_i - \boldsymbol{h}_i^{\mathsf{T}} \boldsymbol{\theta}). \tag{66}$$

As shown by [40], [41], the analytic center achieves a good trade-off in terms of simplicity and practical performance. However, it might not be friendly for analyzing the volume reduction compared to using the center of MVE.

REFERENCES

[1] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, 2020.

[2] M. Kuderer, S. Gulati, and W. Burgard, "Learning driving styles for autonomous vehicles from demonstration," in *International Conference on Robotics and Automation*. IEEE, 2015, pp. 2641–2646.

[3] P. Englert, N. A. Vien, and M. Toussaint, "Inverse kkt: Learning cost functions of manipulation tasks from demonstrations," *The International Journal of Robotics Research*, vol. 36, no. 13-14, pp. 1474–1488, 2017.

[4] W. Jin, T. D. Murphey, D. Kulić, N. Ezer, and S. Mou, "Learning from sparse demonstrations," *IEEE Transactions on Robotics*, pp. 1–20, 2022.

[5] W. Jin, D. Kulić, S. Mou, and S. Hirche, "Inverse optimal control from incomplete trajectory observations," *The International Journal of Robotics Research*, vol. 40, no. 6-7, pp. 848–865, 2021.

[6] A. Jain, S. Sharma, T. Joachims, and A. Saxena, "Learning preferences for manipulation tasks from online coactive feedback," *The International Journal of Robotics Research*, vol. 34, no. 10, pp. 1296–1313, 2015.

[7] A. Bajcsy, D. P. Losey, M. K. O'Malley, and A. D. Dragan, "Learning robot objectives from physical human interaction," *Conference on Robot Learning*, vol. 78, pp. 217–226, 2017.

[8] J. Y. Zhang and A. D. Dragan, "Learning from extrapolated corrections," in *International Conference on Robotics and Automation*. IEEE, 2019, pp. 7034–7040.

[9] D. P. Losey and M. K. O'Malley, "Including uncertainty when learning from human corrections," in *Conference on Robot Learning*, 2018, pp. 123–132.

[10] A. Jain, B. Wojcik, T. Joachims, and A. Saxena, "Learning trajectory preferences for manipulators via iterative improvement," in *Advances in neural information processing systems*, 2013, pp. 575–583.

[11] P. Shivaswamy and T. Joachims, "Online structured prediction via coactive learning," in *International Conference on Machine Learning*, 2012, pp. 59–66.

[12] P. Moylan and B. Anderson, "Nonlinear regulator theory and an inverse optimal control problem," *IEEE Transactions on Automatic Control*, vol. 18, no. 5, pp. 460–465, 1973.

[13] A.-S. Puydupin-Jamin, M. Johnson, and T. Bretl, "A convex approach to inverse optimal control and its application to modeling human locomotion," in *International Conference on Robotics and Automation*, 2012, pp. 531–536.

[14] W. Jin, D. Kulić, J. F.-S. Lin, S. Mou, and S. Hirche, "Inverse optimal control for multiphase cost functions," *IEEE Transactions on Robotics*, vol. 35, no. 6, pp. 1387–1398, 2019.

[15] A. Y. Ng, S. J. Russell *et al.*, "Algorithms for inverse reinforcement learning." in *International Conference on Machine Learning*, vol. 1, 2000, p. 2.

[16] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning." in *Association for the Advancement of Artificial Intelligence*, 2008.

[17] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *International Conference on Machine Learning*, 2006, pp. 729–736.

[18] W. Jin, Z. Wang, Z. Yang, and S. Mou, "Pontryagin differentiable programming: An end-to-end learning and control framework," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[19] Z. Liang, W. Jin, and S. Mou, "An iterative method for inverse optimal control," in *Asian Control Conference*, 2022, pp. 959–964.

[20] C. Moro, G. Nejat, and A. Mihailidis, "Learning and personalizing socially assistive robot behaviors to aid with activities of daily living," *International Conference on Machine Learning*, vol. 7, no. 2, pp. 1–25, 2018.

[21] A. Bobu, A. Bajcsy, J. F. Fisac, and A. D. Dragan, "Learning under misspecified objective spaces," in *Conference on Robot Learning*, 2018, pp. 796–805.

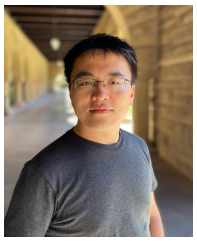
[22] A. D. Dragan, K. Muelling, J. A. Bagnell, and S. S. Srinivasa, "Movement primitives via optimization," in *International Conference on Robotics and Automation*. IEEE, 2015, pp. 2339–2346.

[23] D. P. Losey and M. K. O'Malley, "Learning the correct robot trajectory in real-time from physical human interactions," *ACM Transactions on Human-Robot Interaction*, vol. 9, no. 1, pp. 1–19, 2019.

[24] R. Tedrake, "Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation," *Course Notes for MIT 6.832*, 2022.

[25] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems." in *International Conference on Informatics in Control, Automation and Robotics*, 2004, pp. 222–229.

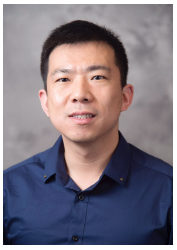
[26] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.
[27] S. Boyd and L. Vandenberghe, "Localization and cutting-plane methods," *Stanford EE 364b Lecture Notes*, 2007.
[28] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
[29] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.
[30] S. P. Tarasov, "The method of inscribed ellipsoids," in *Soviet Mathematics-Doklady*, vol. 37, no. 1, 1988, pp. 226–230.
[31] J. B. Kuipers, *Quaternions and rotation sequences*. Princeton University Press, 1999, vol. 66.
[32] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se(3)," in *IEEE Conference on Decision and Control*, 2010, pp. 5420–5425.
[33] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *International Conference on Machine Learning*, 2017, pp. 136–145.
[34] W. Jin, S. Mou, and G. J. Pappas, "Safe pontryagin differentiable programming," in *Advances in Neural Information Processing Systems*, 2021.
[35] W. Jin, A. Aydinoglu, M. Halm, and M. Posa, "Learning linear complementarity systems," *Learning for Dynamics and Control Conference*, 2022.
[36] D. J. Newman, "Location of the maximum on unimodal surfaces," *Journal of the ACM*, vol. 12, no. 3, pp. 395–398, 1965.
[37] J. Elzinga and T. G. Moore, "A central cutting plane algorithm for the convex programming problem," *Mathematical Programming*, vol. 8, no. 1, pp. 134–145, 1975.
[38] J.-L. Goffin and J.-P. Vial, "On the computation of weighted analytic centers and dual ellipsoids with the projective algorithm," *Mathematical Programming*, vol. 60, no. 1-3, pp. 81–92, 1993.
[39] B. Grünbaum *et al.*, "Partitions of mass-distributions and of convex bodies by hyperplanes." *Pacific Journal of Mathematics*, vol. 10, no. 4, pp. 1257–1261, 1960.
[40] D. S. Atkinson and P. M. Vaidya, "A cutting plane algorithm for convex programming that uses analytic centers," *Mathematical Programming*, vol. 69, no. 1-3, pp. 1–43, 1995.
[41] Y. Nesterov, "Cutting plane algorithms from analytic centers: efficiency estimates," *Mathematical Programming*, vol. 69, no. 1, pp. 149–176, 1995.



Wanxin Jin is a postdoctoral researcher in the GRASP Laboratory at the University of Pennsylvania. He received the Ph.D. degree in Autonomy and Control at Purdue University in 2021. From 2016 to 2017, he was a Research Assistant at Technical University Munich, Germany. Wanxin's research interests include robotics, control, machine learning, and optimization, with emphasis on learning, planning, and control of robots as they interact with the world and humans.

Todd D. Murphey received his B.S. degree in mathematics from the University of Arizona and the Ph.D. degree in Control and Dynamical Systems from the California Institute of Technology. He is a Professor of Mechanical Engineering at Northwestern University. His laboratory is part of the Center for Robotics and Biosystems, and his research interests include robotics, control, machine learning in physical systems, and computational neuroscience.

Zehui Lu received the M.S. degree in mechanical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2019. Between January 2020 and July 2020, he was a research engineer at UM Ford Center for Autonomous Vehicles (FCAV) and ROAHM LAB, all with the University of Michigan. Currently, he is working toward the Ph.D. degree in aeronautics and astronautics engineering at Purdue University, West Lafayette, IN, USA. His current research interests include autonomy, control, and optimization, and their applications in robotics and multi-agent systems.



Shaoshuai Mou is an Associate Professor in the School of Aeronautics and Astronautics at Purdue University. Before joining Purdue, he received a Ph.D. in Electrical Engineering at Yale University in 2014 and worked as a postdoc researcher at MIT for a year after that. His research interests include multi-agent system, control and learning, robotics control, human-robot teaming, resilient autonomy, and also experimental research involving autonomous air and ground vehicles. Dr. Mou co-directs Purdue University's Center for Innovation in Control, Optimization and Networks (ICON), which aims to integrate classical theories in control/optimization/networks with recent advances in machine learning/AI/data science to address fundamental challenges in autonomous and connected systems.