Learning for Control: An Inverse Optimization Approach

Syed Adnan Akhtar, Arman Sharifi Kolarijani and Peyman Mohajerin Esfahani

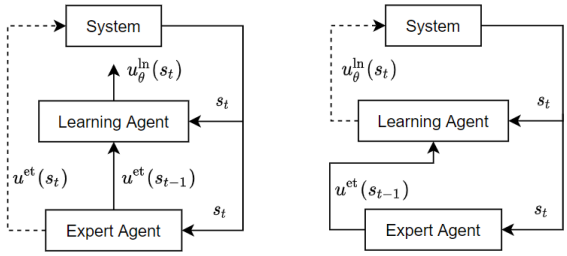
***Abstract*—We present a learning method to learn the mapping from an input space to an action space, which is particularly suitable when the action is an optimal decision with respect to a certain unknown cost function. We use an inverse optimization approach to retrieve the cost function by introducing a new loss function and a new hypothesis class of mappings. A tractable convex reformulation of the learning problem is also presented. The method is effective for learning input-action mapping in continuous input-action space with input-output constraints, typically present in control systems. The learning approach can be effectively transformed to learn a Model Predictive Control (MPC) behaviour and a case study to mimic an MPC is presented, which is a rather computationally heavy control strategy. Simulation and experimental results show the effectiveness of the proposed approach.**

***Index Terms*—Learning-based control, supervised learning, inverse optimization, convex reformulation.**

I. Introduction

REINFORCEMENT learning has gathered interest in the learning community recently [1] where learning of the expert action is based on rewards. Generally, one has access to the expert demonstrations, but not the reward/cost function that dictates the expert action. Imitation Learning involves inferring the optimum policy through expert demonstrations [2] without knowing the reward function. It has been used to teach sequential skills to a robotic arm [3] or acrobatic maneuvers to a helicopter [4].

Numerous methods have been proposed for imitation learning. One of the straightforward methods is to view imitation learning as a supervised learning problem, known as behaviour cloning [5]. Such methods directly learn a mapping from the state space to the action space through expert demonstrations [6]. Alternatively, Inverse Reinforcement Learning (IRL) methods construct an expert policy by retrieving the expert reward function [7]. These methods predominantly follow a Markov decision process framework. See e.g., the maximum margin approach in [8] and the linear programming approach in [9]. An alternative approach is entropy maximization that aims to retrieve a distribution over potential reward functions [10]. In relative entropy methods, the KL-divergence between two trajectories is minimized [11]. Bayesian IRL methods use the state-action pair observations to perform a Bayesian update of a prior distribution over a hypothesis of reward functions [12]. Most of the IRL algorithms are designed for discrete state-action spaces [13]. However, the state-action space in control or robotics is typically continuous and effective discretization leads to exponential growth in the number of states. Consider Figure 1 depicting the nature of such problems in the context of control. There are two settings in Figure 1a and Figure 1b with a difference who (expert or learner) drives the system. The goal of the learning agent is to learn the cost function of the expert regardless of the choice of the setting in Figure 1. At each time step, the expert takes an optimal action $u^{\text{et}}(s_t)$ by solving a parametric optimization problem depending on an exogenous signal s_t. The learner observes the expert action $u^{\text{et}}(s_t)$ with a one time-step delay and infers the cost function that the expert optimizes. Subsequently, the learning agent can mimic the expert action through the learned cost. The learning agent action, denoted by $u^{\text{ln}}_\theta(s_t)$, is in general a suboptimal action since it is generated based on an approximated (or learned) cost rather than the true (or expert) cost. Notice that the true cost is unknown to the learning agent and only available to the expert. In this paper, we focus on learning of the cost function that explains the expert actions possibly in the presence of some state-action constraints.



(a) Expert drives the system. (b) Learning agent drives system.

Fig. 1: Learning settings

An example that can be cast as a learning problem is MPC [14]. Online optimization renders MPC computationally demanding and restricts its application to systems with moderate size dynamics. There are numerous studies in the literature to reduce the computational burden of MPC such as exploiting the structure of the optimization problem [15], *warm-start* approach [16] and *explicit* MPC [17], to name a few. Fundamentally speaking, MPC finds a mapping from the system states to the optimal control inputs. In the context of learning problems, a natural approach to learn this mapping is *supervised learning*. There are several studies that learn (or approximate) the MPC controller in the context of supervised learning either through *indirect learning* [18] or *direct learning* [19]. In the former class, the mapping from the system

The authors are with the Delft Center for Systems & Control, TU Delft, The Netherlands ({S.A.Akhtar, A.SharifiKolarijani, P.MohajerinEsfahani}@tudelft.nl). This research is supported by the ERC grant TRUST-949796.

states to the optimal MPC cost is approximated. Then, the approximated cost is used to obtain a sub-optimal input. In the latter class, the mapping from the states to the optimal input is directly approximated. We emphasize that respecting state and input constraints in supervised learning is generally a challenge, particularly from a computational perspective during the training phase. This computational challenge is at the center of the contribution of this study.

A central example of this study is to learn MPC as an "expert agent." We propose an indirect learning approach based on the inverse optimization [20] that satisfies the input constraints by construction. We refer the reader to the extended version of this paper [21] for an online learning approach to address the limited memory and computational constraints for real systems, as well as a more detailed discussion on the experimental setup and additional numerical examples.

Contributions: In the context explained above, the main contributions of this paper are summarized as follows:

- Inspired by the inverse optimization framework, we introduce parametric optimization as a new hypothesis class along with a loss function that enjoys a tractable reformulation during the training phase (Section III).
- We develop a nonlinear convex reformulation of the target objective function (Theorem 2), as well as a tractable linear matrix inequality (LMI) (Corollary 3).
- We discuss the theoretical results in an MPC setting and how our results help reduce the planning horizon to essentially 1-step. We also implement the proposed learning-based controller in a closed-loop fashion (Section V).

Notations: For a non-negative integer n, $\mathbb{R}^n$ and $\mathbb{R}^n_+$ denote the spaces of n-dimensional reals and non-negative reals, respectively. The identity square matrix with dimension n is denoted by I_n. For a symmetric matrix Q, the inequality $Q \succeq 0$ (respectively, $Q \succ 0$) means that Q is positive semi-definite (respectively, positive definite). Given a vector $x \in \mathbb{R}^n$, we use the shorthand notation $||x||_Q^2 := x^\top Q x$. A symmetric matrix is often described by the upper diagonal elements while the lower diagonal elements is replaced by "$*$". Throughout this study we also reserve the hat notation (e.g., $\widehat{x}$) for the objects dependent on data.

II. Preliminaries

In this section we briefly explain two key problems in the learning literature that are central to objective of this study.

A. Supervised learning

Supervised learning is one of the prospective ways to solve the imitation learning problem [5]. Supervised learning intends to learn an unknown mapping, $h^\star : \mathbb{S} \to \mathbb{U}$, from an input vector $s \in \mathbb{S} \subseteq \mathbb{R}^n$ to an output vector $u \in \mathbb{U} \subseteq \mathbb{R}^m$. Since the space of the candidate function is typically large, we restrict our search to functions within a *hypothesis space* $\mathcal{H}$. A classical example is the collection of all linear functions. We refer to each candidate mapping as a *hypothesis function* h that belongs to the hypothesis space $\mathcal{H}$. The aim is to find a function h that replicates the unknown *ground truth mapping* $h^\star$ as closely as possible. Many algorithms find this hypothesis function h by solving an optimization program that involves a *loss function* $\ell : \mathbb{U} \times \mathbb{U} \to \mathbb{R}_+$. Given a sample (s, u), the loss value $\ell(u, h(s))$ essentially quantifies the mismatch between the predicted output $h(s)$ and the true output u. In supervised learning, a *training* set $\{(\widehat{s}_t, \widehat{u}_t)\}_{t \le T}$ is available where each $(\widehat{s}_t, \widehat{u}_t) \in \mathbb{S} \times \mathbb{U}$ represents an input-output sample, and T denotes the number of samples. Given this dataset, such algorithms solve the so-called *in-sample* error described as

$$\min_{h \in \mathcal{H}} \sum_{t=1}^{T} \ell\left(\widehat{u}_t, h(\widehat{s}_t)\right). \tag{1}$$

A typical hypothesis class is the space of linear functionals

$$\mathcal{H} = \Big\{ h : \mathbb{R}^n \to \mathbb{R}^m \mid h(s) = As,\, A \in \mathbb{R}^{m \times n} \Big\}, \tag{2}$$

where the input and output sets are typically the entire space, i.e., $\mathbb{S} = \mathbb{R}^n$ and $\mathbb{R}^m = \mathbb{U}$. With regards to the loss function, a popular example is the squared 2-norm loss $\ell(u_1, u_2) = \|u_1 - u_2\|_2^2$ where $u_1, u_2 \in \mathbb{U}$. The linear hypothesis class together with the squared 2-norm loss yields a standard regression problem known as the *least squares methods* described through the optimization program

$$\widehat{A}_T^{\text{Reg}} := \operatorname*{arg\,min}_{A \in \mathbb{R}^{m \times n}} \sum_{t=1}^{T} \left\| \widehat{u}_t - A\widehat{s}_t \right\|_2^2. \tag{3}$$

While the least squares method (3) is a powerful estimation tool, it is however not applicable in cases where the output set is a strict subset $\mathbb{U} \subsetneq \mathbb{R}^m$. One may impose such constraints explicitly via, for instance, a projection operator $\Pi_{\mathbb{U}}$. This alters the training program to $\min_A \sum_{t=1}^{T} \|\widehat{u}_t - \Pi_{\mathbb{U}}(A\widehat{s}_t)\|_2^2$. However, this modified training objective is unfortunately no longer convex in the model parameter A. Therefore, constraint satisfaction is a challenge with classical methods in the supervised learning literature.

B. Inverse optimization

Inverse optimization aims to learn the behavior of a decision-maker whose decisions may be influenced by an exogenous environmental signal. More specifically, it is believed that the decision-maker upon receiving a signal $s \in \mathbb{S} \subset \mathbb{R}^n$ optimizes an unknown objective function $u \mapsto F^\star(s, u)$ over a feasible set of actions $\mathbb{U}(s)$, which may also depend on the signal s. In the context of the learning problem depicted in Figure 1b, the signal s and the decision-maker may be seen as the state of the dynamical system and the expert agent, respectively. With this in mind, we hereafter refer to the decisions optimizing the objective $F^\star$ by $u^{\text{et}}(s)$. For ease of notation, we will often omit the dependency of u^{et} on s. Therefore, the inverse optimizing problem is described via the forward optimization program

$$u^{\text{et}}(s) := \operatorname*{arg\,min}_{u \in \mathbb{U}(s)} F^\star(s, u). \tag{4}$$

Recall the mission of the learning agent in Figure 1b; it aims to replicate the behavior of the expert agent. One can approach this objective through the lens of supervised learning. However, as pointed out earlier in Section II.A, the usual

spaces such as the linear hypothesis (2) do not necessarily respect the decision constraint set $u^{\text{et}} \in \mathbb{U}(s)$.

Alternatively, the learning agent can aim to learn the unknown objective function $F^\star$ in (4). To this end, a hypothesis space can be a collection of parameterized functions $\mathcal{F} = \{F_\theta : \mathbb{S} \times \mathbb{U} \to \mathbb{R} \mid \theta \in \Theta\}$ where $\mathbb{U} \supset \mathbb{U}(s)$ denotes a superset of all admissible decisions and $\theta \in \Theta$ represents the parameter to be learnt. The mapping $\theta \mapsto F_\theta$ and the choice of space Θ depend on the problem at hand. In contrast with supervised learning in Section II.A, the input and output sets are now considered as $\mathbb{S} \times \mathbb{U}$ and $\mathbb{R}$, respectively. It is worth noting that in this perspective, the difference on the formal definition of the input and output sets has an important consequence: The training data should now constitute the triple $((s, u^{\text{et}}), F^\star(s, u^{\text{et}}))$.

An approach bridging these two perspectives mentioned above is to utilize parametric objective functions $F_\theta \in \mathcal{F}$ and define a hypothesis space $\mathcal{H}$ containing functions from $s \in \mathbb{S}$ directly to $u \in \mathbb{U}$. More specifically, the $\arg\min$ functions

$$u_\theta^{\text{ln}}(s) = h_\theta(s) := \underset{u \in \mathbb{U}(s)}{\arg\min}\, F_\theta(s, u), \tag{5}$$

can be a natural basis to predict the experts behavior. Notice that the hypothesis candidate $u_\theta^{\text{ln}}(s)$ respects the constraints $u_\theta^{\text{ln}}(s) \in \mathbb{U}(s)$, for all $s \in \mathbb{S}$, by construction. Now given T observations $\{(\widehat{s}_t, \widehat{u}_t^{\text{et}})\}_{t \le T}$ and a loss function $\ell : \mathbb{U} \times \mathbb{U} \to \mathbb{R}_+$, the training procedure (1) is

$$\min_{\theta \in \Theta} \sum_{t=1}^{T} \ell\Big(\widehat{u}_t^{\text{et}}, u_\theta^{\text{ln}}(\widehat{s}_t)\Big). \tag{6}$$

We emphasize that the tractability of (6) highly depends on the set $\mathcal{F}$, more specifically the mapping $\theta \mapsto u_\theta^{\text{ln}}(s)$, and the loss function ℓ. We focus on this in the next section.

III. Proposed Learning Approach

The aim of this section is to elaborate on the choice of the hypothesis space $\mathcal{F}$ described in the previous section and the loss function ℓ to make the training procedure (6) efficient.

A. Hypothesis class

As a particular example of $\mathcal{F}$, we consider a family of quadratic functions defined as

$$\mathcal{F} = \left\{ F_\theta(s, u) = \begin{bmatrix} s \\ u \end{bmatrix}^\top \theta \begin{bmatrix} s \\ u \end{bmatrix} \;\middle|\; \theta \in \Theta \right\}, \tag{7}$$

where Θ is a subset of square matrices $\mathbb{R}^{(m+n)\times(m+n)}$. We can then introduce the following hypothesis class, that is as a collection of mappings $h_\theta : \mathbb{R}^n \to \mathbb{R}^m$,

$$\mathcal{H} = \left\{ h_\theta(s) = \underset{u \in \mathbb{U}(s)}{\arg\min} \begin{bmatrix} s \\ u \end{bmatrix}^\top \theta \begin{bmatrix} s \\ u \end{bmatrix} \;\middle|\; \theta \in \Theta \right\}. \tag{8}$$

Similar hypothesis classes have been actually studied in the literature in the context of continuous time, infinite horizon, but unconstrained optimal control problems [22]. Next, we discuss the choice of the set Θ. Let us denote

$$\theta = \begin{bmatrix} \theta_{ss} & \theta_{su} \\ \theta_{us} & \theta_{uu} \end{bmatrix}. \tag{9}$$

Considering that the ultimate goal is to replicate the expert action, the critical entity is the hypothesis $h_\theta \in \mathcal{H}$ defined in (8). In this view, it is straightforward to observe that the element θ_{ss} in (9) does not play any role in the behavior of h_θ. Moreover, in order to guarantee that the hypothesis h_θ is a computationally tractable oracle, i.e., it is a convex optimization, it is also required to ensure that $\theta_{uu} \succeq 0$. These observations, together with the fact that scaling the function F_θ with a positive scalar also does not have any impact on h_θ, leads us to introduce the set

$$\Theta = \left\{ \theta = \begin{bmatrix} 0 & \theta_{su} \\ \theta_{su}^\top & \theta_{uu} \end{bmatrix} \;\middle|\; \theta_{uu} \succeq I_m \right\}. \tag{10}$$

B. Loss function

A loss function $\ell : \mathbb{U} \times \mathbb{U} \to \mathbb{R}_+$ quantifies the inaccuracy of a hypothesis $h_\theta \in \mathcal{H}$. One can borrow the classical squared 2-norm loss as in the least squares method and define the predictability loss [20] as

$$\ell^{\text{pr}}\Big(u^{\text{et}}(s), u_\theta^{\text{ln}}(s)\Big) := \|u^{\text{et}}(s) - u_\theta^{\text{ln}}(s)\|_2^2, \tag{11}$$

where the learning agent action $u_\theta^{\text{ln}}(s)$ is as defined in (5). The above loss function has a clear interpretation in the context of inverse optimization: It penalizes the error between the decisions of the expert and the learning agent. Despite such a useful interpretation, it is unfortunately shown that the mapping $\theta \mapsto \ell^{\text{pr}}(u^{\text{et}}, u^{\text{ln}}(s))$ is non-convex [23].

In this study, we utilize a rather unconventional loss function in the context of supervised learning. This loss function is particularly suitable for the class of inverse optimization problems where the observed data consists of optimal decisions. Unlike the classical loss functions, the proposed loss function, which we name suboptimality loss, penalizes the mismatch between the expert and learning agent actions "nonuniformly". Let us define the suboptimality loss $\ell^{\text{sub}} : \mathbb{S} \times \mathbb{U} \times \mathbb{U} \to \mathbb{R}_+$ as

$$\begin{aligned} \ell^{\text{sub}}(s, u^{\text{et}}, u_\theta^{\text{ln}}) &:= F_\theta(s, u^{\text{et}}) - F_\theta(s, u_\theta^{\text{ln}}) \\ &= F_\theta(s, u^{\text{et}}) - \min_{u \in \mathbb{U}(s)} F_\theta(s, u). \end{aligned} \tag{12}$$

The loss function (12) effectively quantifies the mismatch between the decisions in terms of their suboptimality level in the candidate hypothesis.

Remark 1 (Regret loss)**.** *The suboptimality loss* (12) *conceptually shares some interesting similarities with the well studied notion of regret loss [24], however, they are different in essence. The regret loss is introduced to measure the performance of optimizing a sequential loss in the form of* (6)*. There are numerous techniques in the online optimization literature in which the decision variable (*θ *in* (6)*) are updated sequentially upon arrival of each data at time* t*. One can indeed resort to these techniques to solve* (6) *when* ℓ *is the suboptimality loss* (12)*.*

Intuitively, suboptimality loss minimization searches for the hypothesis function in the hypothesis space that best explains the expert action u^{et} given an external input s immaterial of the true cost incurred by the agent. Notice that the loss goes to zero

only when the expert action u^{et} is indeed the minimizer of the hypothesized cost. Notice also that the candidate hypothesis depends on the exogenous signal s. Thus, "suboptimality" is attributed to this loss. As opposed to usual loss functions in supervised learning (e.g., the predictability loss ℓ^{pr} in (11)), the suboptimality loss depends explicitly on the signal s. Given the loss function (12) and a dataset $\{(\widehat{s}_t, \widehat{u}_t^{\text{et}})\}_{t\leq T}$, the training phase of the inverse optimization approach yields the optimization program

$$\widehat{\theta}_T^{\text{Inv}} = \operatorname*{argmin}_{\theta\in\Theta}\Big\{ \sum_{t=1}^{T} \Big(F_\theta(\widehat{s}_t, \widehat{u}_t^{\text{et}}) - \min_{u_t\in\mathbb{U}(\widehat{s}_t)} F_\theta(\widehat{s}_t, u_t) \Big) \Big\}. \quad (13)$$

The key computational feature of (12) is that the loss function is convex in θ when the mapping $\theta \mapsto F_\theta$ is linear (e.g., the hypothesis class (7)), a feature missing in the case of the predictability loss (11). To see this, it suffices to notice that the function $\theta \mapsto \ell^{\text{sub}}(s, u^{\text{et}}, u_\theta^{\text{In}}(s))$ constitutes a pointwise maximum of linear functions.

Recall from Section II.B that an alternative (indirect) approach to learn expert action described in (4) is through learning the unknown objective function $F^\star$. This viewpoint considers the set $\mathcal{F}$ as the main hypothesis space, in which the learning phase requires access to a dataset $\{(\widehat{s}_t, \widehat{u}_t^{\text{et}}), F^\star(\widehat{s}_t, \widehat{u}_t^{\text{et}})\}_{t\leq T}$, i.e., it requires additional information $\{F^\star(\widehat{s}_t, \widehat{u}_t^{\text{et}})\}_{t\leq T}$. In such a setting, one can cast the learning problem as a standard regression problem akin to (3). This leads to the optimization program

$$\widehat{\theta}_T^{\text{Reg}} = \operatorname*{argmin}_{\theta\in\Theta}\Big\{ \sum_{t=1}^{T} \Big\| F_\theta(\widehat{s}_t, \widehat{u}_t^{\text{et}}) - F^\star(\widehat{s}_t, \widehat{u}_t^{\text{et}}) \Big\|_2^2 \Big\}, \quad (14)$$

where F_θ has the quadratic form defined in (7) with the feasible set Θ defined in (10).

C. Tractable Reformulation

We now show how the optimization program (6) emerging from the training phase of the inverse optimization approach can be solved efficiently. Note that the optimization (6) is essentially a robust program, i.e., a minimization over the cost parameter $\theta \in \Theta$ and then maximization over $u_t \in \mathbb{U}(\widehat{s}_t)$.

Theorem 2 (Convex reformulation)**.** *Consider the optimization problem (13) with suboptimality loss (12) where the candidate function F_θ admits quadratic form as in (7) and $\mathbb{U}(s) = \{u \in \mathbb{R}^m : M(s)u \leq W(s)\}$, where the parametric matrices $M(s) \in \mathbb{R}^{d\times m}$ and $W(s) \in \mathbb{R}^d$ are given for any admissible signal s. Then, the program (13) is equivalent to*

$$\begin{aligned} \min_{\substack{\theta\in\Theta \\ \lambda_t\geq 0}} \sum_{t=1}^{\top} & F_\theta(\widehat{s}_t, \widehat{u}_t^{et}) + \tfrac{1}{4}\left\| M(\widehat{s}_t)^\top \lambda_t + 2\theta_{su}^\top \widehat{s}_t \right\|_{\theta_{uu}^{-1}}^2 \\ & + W(\widehat{s}_t)^\top \lambda_t, \end{aligned} \quad (15)$$

where θ is as in (9) and $\lambda_t \in \mathbb{R}_+^d$ is the Lagrange multiplier.

Proof. As the main building block, we first reformulate

$$\min_v \Big\{ F_\theta(\widehat{s}_t, v) \;:\; M(\widehat{s}_t)v \leq W(\widehat{s}_t) \Big\}. \quad (16)$$

where v is an-$\mathbb{R}^m$ vector and represents the learner action. The matrices M and W encode the input-output constraints. For ease of notation, we omit writing the dependency of the matrices M and W on $\widehat{s}_t$. Define the Lagrangian function

$$\begin{aligned} L(\lambda_t, v) &= F_\theta(\widehat{s}_t, v) + (Mv - W)^\top \lambda_t \\ &= v^\top \theta_{uu} v + (2\theta_{su}^\top \widehat{s}_t + M^\top \lambda_t)^\top v - W^\top \lambda_t. \end{aligned}$$

The dual function is defined as $g(\lambda_t) = \inf_v L(\lambda_t, v)$. To find the optimal v^*, we set $\nabla_v L(\lambda_t, v) = 0$. Hence,

$$\nabla_v L(\lambda_t, v) = 2\theta_{uu} v + 2\theta_{xu}^\top \widehat{s}_t + M^\top \lambda_t = 0,$$

and as a result, $v^* = -\frac{1}{2}\,\theta_{uu}^{-1}(M^\top \lambda_t + 2\theta_{su}^\top \widehat{s}_t)$. We now substitute v^* in the dual function $g(\lambda_t)$ and arrive at

$$g(\lambda_t) = -\frac{1}{4}\|M^\top \lambda_t + 2\theta_{su}^\top \widehat{s}_t\|_{\theta_{uu}^{-1}}^2 - W^\top \lambda_t.$$

Observe that

$$\max_{\lambda_t\geq 0} g(\lambda_t) = -\min_{\lambda_t\geq 0} \left\{ \frac{1}{4}\|M^\top \lambda_t + 2\theta_{su}^\top \widehat{s}_t\|_{\theta_{uu}^{-1}}^2 + W^\top \lambda_t \right\}.$$

The above equality holds because the program (16) has a quadratic convex cost with affine constraints, which implies strong duality (Slater's condition); and its RHS is equivalent to the program (16). Next, we reformulate (6) by using the above observation. This yields

$$\min_{\theta\in\Theta} \sum_{t=1}^{T} \left(F_\theta(\widehat{s}_t, \widehat{u}_t^{\text{et}}) + \min_{\lambda_t\geq 0} \left\{ \frac{1}{4}\|M^\top \lambda_t + 2\theta_{xu}^\top \widehat{s}_t\|_{\theta_{uu}^{-1}}^2 + W^\top \lambda_t \right\} \right).$$

Moving $\min_{\lambda_t\geq 0}$ outside the sum concludes the proof. □

While the program (15) is convex, it does not follow any particular structure and one has to resort to generic-purpose convex optimization solver for numerical purposes. Next, we show that the program (15) can be translated into a subclass of convex optimization known as the LMI, which is amenable to tailored efficient off-the-shelf solvers like MOSEK [25].

Corollary 3 (LMI reformulation)**.** *The optimization problem (15) admits the LMI reformulation*

$$\left\{ \begin{array}{ll} \min & \sum_{t=1}^{T} \left(F_\theta(\widehat{s}_t, \widehat{u}_t^{et}) + \frac{1}{4}\gamma_t + W(\widehat{s}_t)^\top \lambda_t \right) \\ \text{s.t.} & \theta \in \Theta \text{ in (10)},\ \lambda_t \in \mathbb{R}_+^d,\ \gamma_t \in \mathbb{R}, \quad \forall t \leq T \\ & \begin{bmatrix} \theta_{uu} & M(\widehat{s}_t)^\top \lambda_t + 2\theta_{su}^\top \widehat{s}_t \\ * & \gamma_t \end{bmatrix} \succeq 0, \ \forall t \leq T \end{array} \right. \quad (17)$$

Proof. In (15), replace $\|M(\widehat{s}_t)^\top \lambda_t + 2\theta_{su}^\top \widehat{s}_t\|_{\theta_{uu}^{-1}}^2$ with an upper-bound γ_t for all $t \leq T$. We get

$$\gamma_t - (M(\widehat{s}_t)^\top \lambda_t + 2\theta_{su}^\top \widehat{s}_t)^\top \theta_{uu}^{-1} (M(\widehat{s}_t)^\top \lambda_t + 2\theta_{su}^\top \widehat{s}_t) \geq 0.$$

We now employ the Schur complement approach. Since $\theta_{uu} \succ 0$, the above inequality holds if and only if

$$\begin{bmatrix} \theta_{uu} & M(\widehat{s}_t)^\top \lambda_t + 2\theta_{su}^\top \widehat{s}_t \\ * & \gamma_t \end{bmatrix} \succeq 0.$$

The desired claim then follows. □

We emphasize that the optimization problem (17) is only required to be solved when (one wants to use extra available information for a better estimate of the cost function) we intend to improve the cost function F_θ, and not necessarily at every time instance.

IV. Case Study: Model Predictive Control

We now use the proposed approach to approximate the *value function* of an MPC problem. Notice that an MPC problem is a *forward* optimization problem [14]. The value function is determined implicitly as a solution to a constrained program. However, it is difficult in general to provide a closed-form representation for the value function.

Consider the linear time-invariant system

$$x_{t+1} = Ax_t + Bu_t, \tag{18}$$

where $x \in \mathbb{X} \subseteq \mathbb{R}^n$ and $u \in \mathbb{U}(x) \subseteq \mathbb{R}^m$ denote the states and inputs of the system, respectively. $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$ denote the system matrices. Assume that sets $\mathbb{X}$ and $\mathbb{U}(x)$ are polytopic and contain the origin. Let N be the horizon length and denote $\boldsymbol{u}_t := (u_t, \dots, u_{t+N-1})$. Define the stage cost $c(x,u) := \|x\|_Q^2 + \|u\|_R^2$ for some matrices $Q \succeq 0$ and $R \succeq 0$. Finally, let the MPC cost be

$$V_N(x_t, \boldsymbol{u}_t) := \sum_{i=0}^{N-1} c(x_{t+i}, u_{t+i}) + V_f(x_{t+N}), \tag{19}$$

where $V_f : \mathbb{R}^n \to \mathbb{R}_+$ represents the terminal cost. Given an initial state x_t, we solve the following MPC problem

$$\begin{array}{lll} \min & V_N(x_t, \boldsymbol{u}_t) & \\ \text{s.t.} & x_{t+i+1} = Ax_{t+i} + Bu_{t+i}, & i = 0, \dots, N-1 \\ & u_{t+i} \in \mathbb{U}(x_{t+i}), & i = 0, \dots, N-1 \\ & x_{t+i} \in \mathbb{X}, & i = 1, \dots, N, \end{array} \tag{20}$$

to obtain $V_N^\star(x_t)$ and an optimal input sequence $\boldsymbol{u}_t^\star$. However, we only apply the first input $u_t^\star$ of the sequence $\boldsymbol{u}_t^\star$ and repeatedly solve the problem (20) at each sampling instance.

Our goal is to use the tools developed in this paper to approximate the value function $V_N^\star(x_t)$ such that the computation of the control action $u_t^\star$ is made lighter w.r.t. (20). In doing so, we employ the optimality condition in dynamic programming, and rewrite the problem (20) as

$$\begin{array}{ll} \min & c(x_t, u_t) + F^\star(x_t, u_t) \\ \text{s.t.} & u_t \in \mathbb{U}(x_t), \end{array} \tag{21}$$

where the *tail* cost $F^\star(x_t, u_t)$ is defined as

$$\begin{array}{lll} \min & \sum_{i=1}^{N-1} c(x_{t+i}, u_{t+i}) + V_f(x_{t+N}) & \\ \text{s.t.} & x_{t+i+1} = Ax_{t+i} + Bu_{t+i}, & i = 1, \dots, N-1 \\ & u_{t+i} \in \mathbb{U}(x_{t+i}), & i = 1, \dots, N-1 \\ & x_{t+i} \in \mathbb{X}, & i = 1, \dots, N. \end{array}$$

In view of (21) and following an indirect learning mindset in the previous section, our main goal is to learn the *tail* cost $F^\star : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}_+$. To fit in the reference tracking framework, we define s_t as a feature vector, composed of a combination of states, x_t and reference signals r_t^x, i.e., we introduce $s_t = [x_t^\top,\ (x_t - r_t^x)^\top,\ r_t^{x\top}]^\top$. We represent the data collected with MPC reference tracking by $\{(\widehat{s}_t, \widehat{u}_t^{\text{et}})\}_{t\le T}$. Notice that approximating the MPC cost function through suboptimality loss does not require knowledge of the true cost value, MPC parameters (Q, R) or the system matrices. In the next section, we present simulation results of mimicking an MPC controller for a lab helicopter.

V. Results

In the previous section, we discussed typecasting the learning problem in an MPC framework. We now discuss empirical results with an experimental setup of a 1-DOF lab helicopter. Recall that the goal through inverse optimization is to approximate the true but unknown cost function that explains the mapping from the system states to the actions for reference tracking. In the context of MPC, the hope is to reduce the computational complexity. For shortage of space, we only present brief results here. A more detailed discussion of the experimental setting as well as additional simulation results on high-dimensional dynamics of a shell heavy oil fractionator is presented in the extended version [21].

In this section, the performance of the learning agent that is trained with the two methods, namely, regression as in (14), and inverse optimization as in (13) is compared. For the comparison, we use the 2-norm of the control input error relative to MPC as a performance metric. The MPC is taken for a prediction horizon of $N = 75$. We will also present the reference tracking error for inverse optimization without expert in the loop, and compare it with that of MPC.

Consider Figure 1b where the learning agent is driving the system and the expert gives corrective advice to the learner in the form of expert actions. At each time t, the learning agent has an estimate F_θ of the true cost $F^\star$ that guides its action. The learner reads the state s_t, and takes an action $u_\theta^{\text{ln}}(s_t)$. Subsequently, the expert (MPC) reveals its action $u^{\text{et}}(s_t)$ (corrective advice). Now, with the new information, $u^{\text{et}}(s_t)$ gained by the learner, it improves its estimate of the true cost function. Therefore, the learning agent decides an action in response to the signal $\widehat{s}_t$ by using the past data upto time $(t-1)$, i.e. $(\widehat{s}_k, \widehat{u}_k^{\text{et}}, F^\star(\widehat{s}_k, \widehat{u}_t^{\text{et}}))$ for all $k = 1, ..., (t-1)$ in addition to $\widehat{s}_t$. We denote this action with $u_{\theta_{t-1}}^{\text{ln}}(\widehat{s}_t)$, obtained by solving the following optimization problem similar to (5)

$$u_{\theta_{t-1}}^{\text{ln}}(\widehat{s}_t) = \underset{u\in\mathbb{U}(\widehat{s}_t)}{\arg\min}\, F_{\theta_{t-1}}(\widehat{s}_t, u), \tag{22}$$

where $\theta_t = \theta_t^{\text{Reg}}$ for regression, $\theta_t = \theta_t^{\text{Inv}}$ for inverse optimization with suboptimality loss. It is worth noting that the learner action at time step t uses θ_{t-1} since the the expert action $\widehat{u}_t^{\text{et}}$ is revealed to the learner in one time step delay. Henceforth, the superscript 'Inv' will be referred to the case where inverse optimization using suboptimality loss is used with the LMI reformulation shown in (17) and 'Reg' will similarly denote the training through regression according to (14). We do not solve the LMI in (17) at each time instance, but only in the simulation for the first $50s$ to study the input error behaviour. An online learning approach to update F_θ at each time instance is presented in the extended version of this paper [21]. Recall that the end goal for the learner was to mimic the expert action, $\widehat{u}_t^{\text{et}}$, which is the control input. Since the expert action $\widehat{u}_t^{\text{et}}$ is not immediately available to the learner, it is a good performance metric to measure the action discrepancy $\|u_{\theta_{t-1}}^{\text{ln}}(\widehat{s}_t) - \widehat{u}_t^{\text{et}}\|_2$.

Consider the first scenario as depicted in Figure 1b for tracking of square-wave of amplitude 0.2, while also receiving corrective advice from the expert at each time instance, for the duration of $T = 200s$. The control input error for such

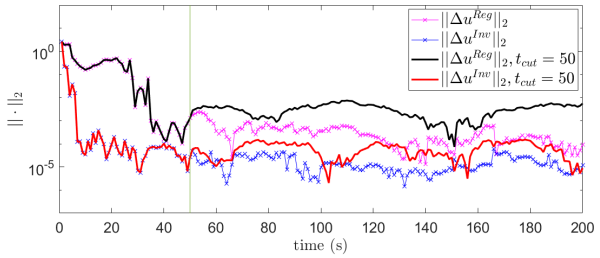


Fig. 2: Simulation results: control input error

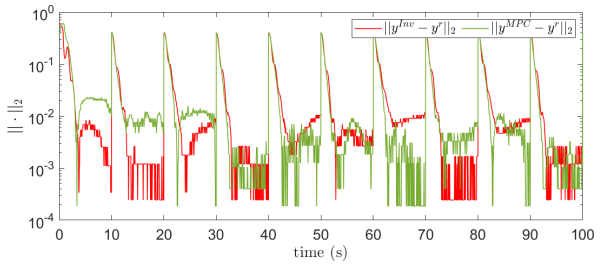


Fig. 3: Experimental results: tracking error

a scenario is shown in Figure 2 for regression (magenta) and inverse optimization (blue). The control input error with inverse optimization drops to almost 10^{-4} in just $10s$ and to 10^{-5} in about $100s$. For regression, it takes about $120s$ for the error to drop to 10^{-4}. Now consider a second scenario as in Figure 1b where the expert (thus its corrective advice) is only available up till time $t = t_{cut}$. Beyond the time $t > t_{cut}$ the expert is removed from the control loop and the learner can no longer improve its estimate F_θ of the true cost function $F^\star$. Therefore the cost F_θ learned up till $t = t_{cut}$ becomes static for the subsequent times $t > t_{cut}$. For such a scenario, the 2-norm of the control input error is presented for regression (black) and inverse optimization (red) in Figure 2 with $t_{cut} = 50s$. It can be observed that until time $t = 50$, the control input errors for both the methods are identical to the previous scenario when the expert was present throughout. However, for $t > t_{cut}$, the error slightly increases after the MPC is removed from the loop, and the static cost of the learner is used for reference tracking of square wave of amplitude 0.2. Therefore, the learner solves a simple quadratic program as in (5) to generate a suboptimal input rather than the computationally heavy MPC. Figure 3 shows the 2-norm of the tracking error with MPC (green) and inverse optimization (red). y^{Inv} is the system output due to the control input u^{Inv} whereas y^{MPC} is the system output due to the MPC inputs $u^{MPC} = u^{\text{et}}$. Occasionally, it can be observed that inverse optimization has lower tracking error than MPC. Finally, we compare the computation time for the expert (MPC) vs the learner. For each control input computation and in the average, the learner takes 0.12 ms against 1.69 ms for MPC. As such, the learning agent is roughly twelve times faster than MPC.

REFERENCES

[1] C. Celemin, G. Maeda, J. R. del Solar, J. Peters, and J. Kober, "Reinforcement learning of motor skills using policy search and human corrective advice," *The International Journal of Robotics Research*, vol. 38, no. 14, pp. 1560–1580, 2019.

[2] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *Foundations and Trends in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.

[3] S. Manschitz, J. Kober, M. Gienger, and J. Peters, "Learning movement primitive attractor goals and sequential skills from kinesthetic demonstrations," *Robotics and Autonomous Systems*, vol. 74, 2015.

[4] A. Coates, P. Abbeel, and A. Y. Ng, "Learning for control from multiple demonstrations," in *Proc. International Conference on Machine Learning*, p. 144–151, 2008.

[5] E. Klein, M. Geist, B. Piot, and O. Pietquin, "Inverse reinforcement learning through structured classification," in *Proc. Advances in Neural Information Processing Systems*, vol. 25, pp. 1007–1015, 2012.

[6] R. Amit and M. Matari, "Learning movement sequences from demonstration," in *International Conference on Development and Learning*, pp. 203–208, 2002.

[7] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proc. International Conference on Machine Learning*, pp. 1–8, 2004.

[8] N. D. Ratliff, D. Silver, and J. A. Bagnell, "Learning to search: Functional gradient techniques for imitation learning," *Autonomous Robots*, vol. 27, no. 1, pp. 25–53, 2009.

[9] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proc. International Conference on Machine Learning*, p. 663–670, 2000.

[10] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning.," in *Proc. Association for the Advancement of Artificial Intelligence*, vol. 8, pp. 1433–1438, 2008.

[11] A. Boularias, J. Kober, and J. Peters, "Relative entropy inverse reinforcement learning," in *Proc. International Conference on Artificial Intelligence and Statistics*, pp. 182–189, 2011.

[12] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning.," in *Proc. International Conference on Artificial Intelligence and Statistics*, vol. 7, pp. 2586–2591, 2007.

[13] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *preprint arXiv:1806.06877*, 2018.

[14] M. Morari and J. H. Lee, "Model predictive control: Past, present and future," *Computers & Chemical Engineering*, vol. 23, no. 4-5, pp. 667–682, 1999.

[15] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on control systems technology*, vol. 18, no. 2, pp. 267–278, 2009.

[16] E. John and A. Yildirim, "Implementation of warm-start strategies in interior-point methods for linear programming in fixed dimension," *Computational Optimization and Applications*, pp. 151–183, 2008.

[17] A. Alessio and A. Bemporad, "A survey on explicit model predictive control," in *Nonlinear model predictive control*, pp. 345–369, Springer, 2009.

[18] M. Zhong, M. Johnson, Y. Tassa, T. Erez, and E. Todorov, "Value function approximation and model predictive control," in *Proc. IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pp. 100–107, 2013.

[19] T. Parisini and R. Zoppoli, "A receding-horizon regulator for nonlinear systems and a neural approximation," *Automatica*, vol. 31, no. 10, pp. 1443 – 1451, 1995.

[20] P. Mohajerin Esfahani, S. Shafieezadeh-Abadeh, G. A. Hanasusanto, and D. Kuhn, "Data-driven inverse optimization with imperfect information," *Mathematical Programming*, vol. 167, pp. 191–234, 2018.

[21] S. A. Akhtar, A. S. Kolarijani, and P. Mohajerin Esfahani, "Learning for control: An inverse optimization approach," 2020. extended version available at http://www.dcsc.tudelft.nl/∼mohajerin/drafts/Lear4C.pdf.

[22] K. G. Vamvoudakis, "Q-learning for continuous-time linear systems: A model-free infinite horizon optimal control approach," *Systems & Control Letters*, vol. 100, pp. 14–20, 2017.

[23] D. Bertsimas, V. Gupta, and I. C. Paschalidis, "Data-driven estimation in equilibrium using inverse optimization," *Mathematical Programming*, vol. 153, no. 2, pp. 595–633, 2015.

[24] E. Hazan, "Introduction to online convex optimization," *Foundations and Trends® in Optimization*, vol. 2, no. 3-4, pp. 157–325, 2016.

[25] M. ApS, *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.*, 2019.