2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Imitation Learning from Nonlinear MPC via the Exact Q-Loss and its Gauss-Newton Approximation

Andrea Ghezzi*[,1], Jasper Hoffman*[,2], Jonathan Frey[1,3], Joschka Boedecker[2], Moritz Diehl[1,3]

***Abstract*— This work presents a novel loss function for learning nonlinear Model Predictive Control policies via Imitation Learning. Standard approaches to Imitation Learning neglect information about the expert and generally adopt a loss function based on the distance between expert and learned controls. In this work, we present a loss based on the Q-function directly embedding the performance objectives and constraint satisfaction of the associated Optimal Control Problem (OCP). However, training a Neural Network with the Q-loss requires solving the associated OCP for each new sample. To alleviate the computational burden, we derive a second Q-loss based on the Gauss-Newton approximation of the OCP resulting in a faster training time. We validate our losses against Behavioral Cloning, the standard approach to Imitation Learning, on the control of a nonlinear system with constraints. The final results show that the Q-function-based losses significantly reduce the amount of constraint violations while achieving comparable or better closed-loop costs.**

I. Introduction

Model Predictive Control (MPC) is an optimization-based approach for controlling dynamical systems [1]. It is used in many different applications due to the versatility, handling of constraints and stability guarantees. For each new state, MPC computes a control by solving an optimal control problem (OCP), that contains the performance objectives, system dynamics and system properties. Yet, the computational complexity of real time optimization can be a major limitation of the applicability of MPC, when a certain control frequency is necessary or one uses embedded systems with constrained computationally resources.

One approach to tackle this problem is *explicit* MPC. Instead of having an *implicit* control policy that is derived from online optimization, *explicit* MPC computes the control policy beforehand in an offline setting which then can be simply evaluated online. For linear MPC it is possible to represent the state feedback policy by a piece-wise affine function and store the corresponding gain as a look-up table. However, for nonlinear MPC (NMPC) is not possible to exactly represent the policy and the available approaches suffer from approximation errors. Therefore, the common choice is to rely on the implicit online computation of the control policy.

A promising approach related to *explicit* MPC is Imitation Learning (IL). Here, one tries to imitate the behavior of the MPC policy with a parameterized policy like a Neural Network. The main issue of this approach is that we introduce approximation errors while imitating. On the other hand, IL can be used, even in the case of NMPC, to drastically reduce online computational costs. Previous works mainly used methods like Behavioral Cloning (BC) to imitate the MPC policy. However, the loss function used in BC is just a surrogate loss that minimizes the difference between the MPC policy and the learned policy [2]. With such a surrogate loss, we lose all information why the MPC took a control in the first place. Thus, during training, the learned policy gets no feedback in terms of constraint satisfaction and performance objectives.

In this work, instead of falling back to a surrogate loss like in BC, we introduce a new loss formulation that exposes to the policy the inner objective of the MPC during training. For this, we introduce a Q-function which corresponds to the cost at the optimal solution of the associated OCP for a given initial state and fixed initial control.

Specifically, our contributions are twofold:

1) we propose a loss for IL based on the Q-function of the given OCP such that the loss directly embeds the characteristics of the OCP;
2) we introduce a quadratic programming approximation of the Q-function loss to reduce the computational burden and make it more suitable for training Neural Networks.

We compare both proposed loss functions against Behavioral Cloning (BC) on the stabilization of a nonlinear cart-pole system. The policies learned with the Q-function losses achieve a lower cost and significantly less constraint violations compared to the BC policy. This is promising since we expect that the potential performance difference between the proposed losses and the standard one will be even more evident on more complex examples.

Related work: For *explicit* linear MPC, it is possible to have an exact representation of the policy via piece-wise affine functions [3], [4]. Via approximation of such policies, *explicit* MPC has been extended to the nonlinear case [5]. The use of Neural Networks to represent optimal control policies has been investigated in [6] and applied to chemical process control in [7]. Thanks to the success of deep learning, the use of Neural Network for *explicit* MPC has grown, with recent applications in power electronics, building control and robot manipulators [8]–[10]. However, these works are based

* These authors contributed equally, order is alphabetical.
[1] Department of Microsystems Engineering (IMTEK), University of Freiburg, 79110 Freiburg, Germany
[2] Department of Computer Science, University of Freiburg
[3] Department of Mathematics, University of Freiburg
Correspondents: andrea.ghezzi@imtek.uni-freiburg.de, hoffmaja@informatik.uni-freiburg.de
This research was supported by DFG via Research Unit FOR 2401 and project 424107692 and by the EU via ELO-X 953348.

on Behavior Cloning, thus they minimize a surrogate loss function.

Regarding properties and guarantees for learned controllers, for linear MPC, two methods are presented in [11] to verify the closed-loop stability of Neural Network controllers and quantify their worst-case approximation error. For NMPC, in [12] a statistical guarantee on stability and constraint satisfaction is derived via a condition on the approximation error of the learned MPC. In [13], control barrier functions are introduced as a way to transfer safety from the expert to the learned controller and a formal guarantee of input-to-state stability is provided. In [14] a trajectory is not only optimized for costs but also whether the trajectories can be recreated by the learned policy.

A very related line of work, looking at IL from MPC, can be found in [15] and [16], where IL is reformulated by minimizing the control Hamiltonian of a continuous-time OCP formulation. One major difference to our approach, apart from that we look at discrete-time OCPs, is that with the Hamiltonian, one does not resolve the OCP for each new control, but only use the gradient of the cost-to-go function with respect to the state. In order to incorporate information about inequality constraints into this gradient, they introduce log-barrier functions to the OCP formulation.

A. Notation

Given $a \in \mathbb{R}^{n_a}$ and $b \in \mathbb{R}^{n_b}$, we denote the vector $c = [a^\top \ b^\top]^\top$ by $c = (a, b)$. Given $a \in \mathbb{R}^{n_a}$, we denote $\|a\|_W^2 \coloneqq a^\top W a$, for every $W \in \mathbb{R}^{n_a \times n_a}$ a positive definite matrix. With $\mathcal{U}(a, b)$, we denote the uniform distribution with boundaries a, b respectively.

II. Background

In this section, we provide the necessary background of Optimal Control and Imitation Learning.

A. Optimal Control

In this work, we want to approximate NMPC policies, which are defined by the repetitive solution of an OCP. Specifically throughout this paper, we regard the following generic discrete-time OCP

$$\min_{\substack{x_0, u_0, s_0, \dots, \\ u_{N-1}, x_N, s_N}} \quad \sum_{k=0}^{N-1} \tilde{L}(x_k, u_k, s_k) + \tilde{E}(x_N, s_N) \tag{1a}$$

$$\text{s.t.} \quad x_0 = \bar{x}_0, \tag{1b}$$

$$x_{k+1} = f(x_k, u_k), k = 0, \dots, N-1, \tag{1c}$$

$$h(x_k, u_k) \leq s_k, \quad k = 0, \dots, N-1, \tag{1d}$$

$$r(x_N) \leq s_N, \tag{1e}$$

$$s_k \geq 0, \qquad k = 0, \dots, N, \tag{1f}$$

with N shooting intervals. Here, $x_k \in \mathbb{R}^{n_x}$ and $u_k \in \mathbb{R}^{n_u}$ represent the state and the control trajectories which follow the possibly nonlinear system dynamics f in (1c). Inequality (1d) enforces path constraints and (1e) encodes a condition on the system terminal state. We assume the functions L, E, f, h, r to be twice continuously differentiable in their respective variables. In order to guarantee feasibility, we introduce slack variables $s_k \in \mathbb{R}^{n_{s,k}}$ and we penalize their use in the cost function. Thus, the stage cost is defined as $\tilde{L}(x_k, u_k, s_k) \coloneqq L(x_k, u_k) + z^\top s_k + \|s_k\|_Z^2$ and terminal cost as $\tilde{E}(x_N, s_N) \coloneqq E(x_N) + z_e^\top s_N + \|s_N\|_{Z_e}^2$. Note, that for some values of the positive slack penalties z, Z an exact penalization of the constraints can be achieved [17]. In contexts where constraint satisfaction is critical we can tune the weights of the slacks to favor feasibility over optimality. In an NMPC scheme, the OCP (1) is solved in every control step for a new initial state $\bar{x}_0$ and the optimal control $u_0^\star(\bar{x}_0)$ is applied to the system.

Note that for notational convenience we derive every further OCP formulation by omitting the slack variables.

B. Imitation Learning

We are interested in using Imitation Learning (IL) to imitate the control law derived by solving a discrete-time OCP as described in (1). We can define the expert policy, that we want to imitate, by the first optimal control $\pi^\star(x) \coloneqq u_0^\star(\bar{x}_0)$ of the solution for a given $\bar{x}_0$. We aim to approximate $\pi^\star$ as well as possible by a parameterized policy $\pi(\cdot; w) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$. A parameterized policy could be for example a Neural Network where the parameter $w \in \mathbb{R}^{n_w}$ are the weights of the Neural Network.

In the following, we will give a short introduction to the IL framework. The IL objective can be defined as

$$\mathcal{L}(w) \coloneqq \mathbb{E}_{x \sim \mathcal{D}} \left[\ell \left(x, \pi(\cdot\,; w) \right) \right], \tag{2}$$

where $\mathcal{D}$ is a given state distribution over $\mathbb{R}^{n_x}$ and ℓ the point wise loss function of the policy $\pi(\cdot\,; w)$ for a given state x [2]. The final goal of IL is then to find the optimal combination of parameters $w^\star$ that minimizes the expected loss $\mathcal{L}(w)$:

$$w^\star = \arg\min_w \quad \mathcal{L}(w). \tag{3}$$

In its most general form, IL assumes no prior knowledge about the internal objective of the expert policy. For example the expert could be a human. Thus, methods like Behavior Cloning, replace the internal objective by using a surrogate loss function ℓ that measures the behavioral difference between the policy π and the expert policy $\pi^\star$. Popular choices are the quadratic loss function ℓ_2 defined as

$$\ell_2(x, \pi) \coloneqq \left(\pi(x) - \pi^\star(x) \right)^2, \tag{4}$$

the Huber loss [18], the ℓ_1 loss or the cross-entropy loss in the case of stochastic policies. In this paper, we will use the quadratic loss function ℓ_2 for comparison, which results in the following expected quadratic loss:

$$\mathcal{L}^2(w) \coloneqq \mathbb{E}_{x \sim \mathcal{D}} \left[\ell_2 \left(x, \pi(\cdot\,; w) \right) \right]. \tag{5}$$

Finally, the state distribution $\mathcal{D}$ seen when deploying the expert policy might differ from the learned policy. This is called the covariate shift and can be mitigated by adopting Dagger [19], which samples controls from a mixture of the expert and the current policy while collecting states from rollouts. We use Dagger for all methods compared in this paper.

III. Q-loss for Imitation Learning

In the following, we will see that when the expert policy is the solution of an OCP, we do not necessarily need a surrogate loss.

A. The exact Q-loss

In fact, we can directly define a loss based on the internal objective of the expert, which we call the exact Q-loss. This proposed loss function directly embeds the information contained in the OCP such as its cost and constraints. The main idea is to reuse the OCP formulation (1) and fix the first control u_0 of the OCP by the value returned from the policy, $\bar{u}_0 = \pi(x; w)$. By solving the resulting OCP, we can assign a cost to the policy value $\pi(x; w)$.

Specifically, we reformulate the OCP (1) to expose exclusively the first control. Given $\bar{x}_0$ we define the exact Q-loss by the following "Q-function OCP"

$$
\begin{aligned}
Q(\bar{x}_0, \bar{u}_0) \coloneqq \min_{\substack{x_0, u_0, \dots, \\ u_{N-1}, x_N}} \quad & \sum_{k=0}^{N-1} L(x_k, u_k) + E(x_N) && \text{(6a)} \\
\text{s.t.} \quad & \\
& x_0 - \bar{x}_0 = 0, && \text{(6b)} \\
& u_0 - \bar{u}_0 = 0, && \text{(6c)} \\
& x_{k+1} - f(x_k, u_k) = 0, && \text{(6d)} \\
& h(x_k, u_k) \leq 0,\ k = 1, \dots, N-1, && \text{(6e)} \\
& r(x_N) \leq 0. && \text{(6f)}
\end{aligned}
$$

We remind that for notational convenience we omit the slack variables in the OCP formulation. With the exact Q-loss, the imitation learning objective becomes

$$
\mathcal{L}^{\mathrm{Q}}(w) \coloneqq \mathbb{E}_{x \sim \mathcal{D}} \left[Q(x, \pi(x; w)) \right]. \tag{7}
$$

The name "Q-loss" is motivated from the related concept of Q-functions in reinforcement learning. The gradient of $\mathcal{L}^{\mathrm{Q}}(w)$ is defined as

$$
\nabla_w \mathcal{L}^{\mathrm{Q}}(w) = \mathbb{E}_{x \sim \mathcal{D}} \Big[\nabla_w \pi(x; w)\ \nabla_u Q(x, u)|_{u=\pi(x;w)} \Big]. \tag{8}
$$

Lemma 1: The gradient of the Q-loss is given by the Langrangian multiplier $\bar{\lambda}_u$ corresponding to the constraint (6c) for the optimal solution

$$
\bar{\lambda}_u = \nabla_u Q(x, u)|_{u=\pi(x;w)}. \tag{9}
$$

Proof: This can be derived by looking at the cost-to-go from dynamic programming and the first order necessary condition of optimality, [1, §8.8.3], [20, §3.3.3]. □

Lemma 2: If $\pi^\star(\bar{x}_0)$ is a unique minimizer of (1) then $Q(\bar{x}_0, \bar{u}_0) > Q(\bar{x}_0, \pi^\star(\bar{x}_0))$ for any $\bar{u}_0 \neq \pi^\star(\bar{x}_0)$. Thus, the exact Q-loss penalizes any deviation of $\bar{u}_0$ from $\pi^\star(\bar{x}_0)$.

When looking at (7), an interesting connection to actor-critic algorithms [21] emerges, where the actor is the policy π which is criticized by a parameterized value function Q, called the critic. In this framework, the Q-function derived from the OCP can be seen as the critic. Following this perspective, the gradient of (8) is directly related to deterministic policy gradients [22].

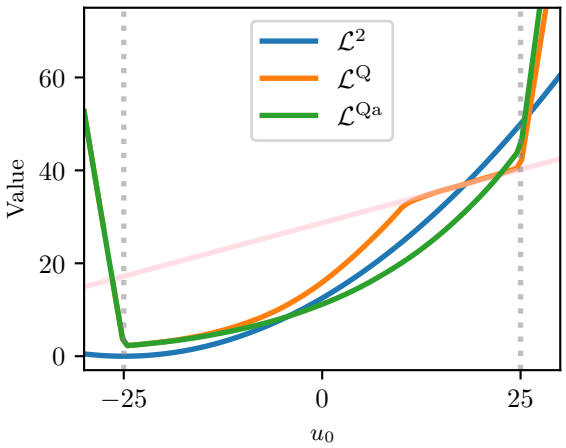

Fig. 1. Comparison of the losses for the example presented in Section IV, for a given initial state $\bar{x}_0 = (0.8, 0, \pi/4, 0)$ and $u_0^\star(\bar{x}_0) = -25$. The pink line corresponds to the gradient of $\mathcal{L}^Q$ at $\bar{u}_0 = 15$.

B. Discussion of the exact Q-loss

The exact Q-loss is computationally more complex than a standard loss. To get a better understanding of the computational costs, one can compare the gradients of behavior cloning and the exact Q-loss. For the behavior cloning loss $\mathcal{L}^2$, the gradient is given by

$$
\nabla_w \mathcal{L}^2(w) = \mathbb{E}_{x \sim \mathcal{D}} \left[\nabla_w \pi(x; w)(\pi(x; w) - \pi^\star(x)) \right]. \tag{10}
$$

Comparing the gradient of both losses, we first note that both require a forward and backward pass of π to compute $\pi(x; w)$ and the gradient of the policy $\nabla_w \pi(x; w)$. The exact Q-loss (8) additionally requires the gradient $\nabla_u Q(x, u)|_{u=\pi(x_i;w)}$, which depends on x and u. Thus, we need to solve the OCP (6) not only for each new sample x but also for each new control u. Note that we get the gradient via the multiplier when solving this OCP (cf. Lemma 1).

Since we might evaluate the loss (7) for any couple $(x, \pi(x; w))$ we need to guarantee feasibility of problem (6) by introducing slack variables to soften the constraint, as presented in problem (1).

Finally, we remark that according to the properties of (6), the function Q might be a piece-wise nonlinear and nonconvex function. As a result of the nonconvexity in u_0, the tangent plane derived from the gradient of Q at a given point $(\bar{x}_0, \bar{u}_0)$ might not be a lower bound for $Q(\bar{x}_0, u_0^\star)$, an example of this is given in Figure 1. This might slow down the minimization of the loss (7).

In the next section, we address the computational complexity of the exact Q-loss issue by proposing a convex approximation of it.

C. The Gauss-Newton Q-loss

We propose a simplified loss which aims to alleviate both the computational burden related to $\mathcal{L}^{\mathrm{Q}}$ and the possible misleading gradients generated by the nonconvexity of Q. The loss exploits the optimal control structure of $\mathcal{L}^{\mathrm{Q}}$ but

it builds a quadratic programming approximation of the Q-function OCP (6) around the optimal solution.

Assumption 1: Let us consider functions L and E in (1a) being the square of vector-valued functions of the form $\|\bar{L}\|^2$ with $\bar{L} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_y}$, $\|\bar{E}\|^2$ with $\bar{E} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_e}$ respectively. Cost functions of this type appear frequently in optimal control since they represent tracking cost, most importantly they allow for a Gauss-Newton Hessian approximation [23].

Given a sample $\bar{x}_0$ and a first control $\bar{u}_0$, we solve (1) and denote its solution as $\zeta = (\tilde{x}_0, \tilde{u}_0, \ldots, \tilde{u}_{N-1}, \tilde{x}_N)$ and we use it as a linearization point for the quadratic approximation of problem (6) as follows

$$
\begin{aligned}
&Q_{\mathrm{a}}(\bar{x}_0, \bar{u}_0) \coloneqq \\
&\min_{\substack{x_0, u_0, \ldots, \\ u_{N-1}, x_N}} \quad \sum_{k=0}^{N-1} \|L_{\mathrm{L}}(x_k, u_k; \tilde{x}_k, \tilde{u}_k)\|^2 + \|E_{\mathrm{L}}(x_N; \tilde{x}_N)\|^2 \\
&\text{s.t.} \qquad x_0 - \bar{x}_0 = 0, \\
&\qquad\quad u_0 - \bar{u}_0 = 0, \\
&\qquad\quad x_{k+1} - f_{\mathrm{L}}(x_k, u_k; \tilde{x}_k, \tilde{u}_k) = 0, \\
&\qquad\quad h_{\mathrm{L}}(x_k, u_k; \tilde{x}_k, \tilde{u}_k) \leq 0,\ k = 1, \ldots, N-1, \\
&\qquad\quad r_{\mathrm{L}}(x_N; \tilde{x}_N) \leq 0,
\end{aligned}
\tag{11}
$$

with $L_{\mathrm{L}}(x_k, u_k; \tilde{x}_k, \tilde{u}_k)$ being defined as the first order Taylor series of L at $(\tilde{x}_k, \tilde{u}_k)$ as follows

$$
\begin{aligned}
L_{\mathrm{L}}(x_k, u_k; \tilde{x}_k, \tilde{u}_k) &= L(\tilde{x}_k, \tilde{u}_k) \\
&\quad + \nabla_{x,u} L(\tilde{x}_k, \tilde{u}_k)^\top \left(\begin{bmatrix} x \\ u \end{bmatrix} - \begin{bmatrix} \tilde{x}_k \\ \tilde{u}_k \end{bmatrix} \right).
\end{aligned}
\tag{12}
$$

The functions $f_{\mathrm{L}}, h_{\mathrm{L}}$ are defined in the same way, while for $E_{\mathrm{L}}, r_{\mathrm{L}}$ the linearization is done at $\tilde{x}_N$ and only with respect to x. The function Q_{a} is now described by a convex piecewise quadratic function.

Lemma 3: If we use the optimal solution of (1) as linearization point, i.e., set $(\bar{\tilde{x}}_0, \bar{\tilde{u}}_0) \coloneqq (\bar{x}_0, \pi^\star(\bar{x}_0))$, then the Gauss-Newton Q-loss is a convex distance function with $Q_{\mathrm{a}}(x, u) \geq Q_{\mathrm{a}}(x, \pi^\star(x))$, for every x, u.

We can introduce the approximate Q-loss as

$$
\mathcal{L}^{\mathrm{Q_a}}(w) \coloneqq \mathbb{E}_{x \sim \mathcal{D}} \left[Q_{\mathrm{a}}(x, \pi(x; w)) \right], \tag{13}
$$

and its gradient $\nabla \mathcal{L}^{\mathrm{Q_a}}(w)$ is given by

$$
\nabla \mathcal{L}^{\mathrm{Q_a}}(w) = \mathbb{E}_{x \sim \mathcal{D}} \left[\nabla_w \pi(x; w)\ \nabla_u Q_{\mathrm{a}}(x, u; \zeta)|_{u = \pi(x; w)} \right]. \tag{14}
$$

Compared to (8) the function $\nabla \mathcal{L}^{\mathrm{Q_a}}$ requires the gradient of the QP problem, i.e., $\nabla_u Q_{\mathrm{a}}$ which is less expensive to compute.

IV. Numerical example

We show the effectiveness of the proposed losses against the standard ℓ_2 loss on the example of the cart-pole. The system is depicted in Figure 2. The task is to control the system such that the rod stays in upright position and the cart stays at the center of the track.

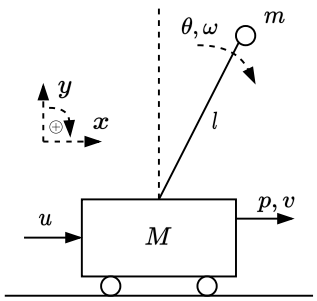

Fig. 2. Schematic of the cart pole

By neglecting friction forces, the dynamics of the system are defined by the following equations

$$
\begin{bmatrix} \dot{p}(t) \\ \dot{v}(t) \\ \dot{\theta}(t) \\ \dot{\omega}(t) \end{bmatrix} = \begin{bmatrix} v \\ \frac{-ml \sin(\theta)\dot{\theta}^2 + mg \cos\theta \sin(\theta) + u}{M + m - m\cos^2(\theta)} \\ \omega \\ \frac{-ml \cos(\theta) \sin(\theta)\dot{\theta}^2 + (M+m) g \sin(\theta) + u \cos(\theta)}{l \cdot (M + m - m \cos^2(\theta))} \end{bmatrix}, \tag{15}
$$

with $l = 0.8\ (\mathrm{m}), m = 0.1\ (\mathrm{kg})$, $M = 1\ (\mathrm{kg})$, $g = 9.81\ (\mathrm{m/s^2})$. The system state is $x(t) = (p(t), v(t), \theta(t), \omega(t)) \in \mathbb{R}^4$, the control is $u(t) \in \mathbb{R}$. We assume full state observability.

We use multiple shooting with $N = 20$ shooting intervals of $\Delta t = 0.05\,(s)$ and an RK4 integrator to discretize (15) and obtain the discrete time OCP

$$
\begin{aligned}
&\min_{\substack{x_0, u_0, \ldots, \\ u_{N-1}, x_N}} \quad \frac{1}{N} \left(\sum_{k=0}^{N-1} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^\top \begin{bmatrix} S & 0 \\ 0 & R \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} + x_N^\top P x_N \right) \\
&\text{s.t.} \qquad x_0 = \bar{x}_0, \\
&\qquad\quad x_{k+1} = f(x_k, u_k), \quad k = 0, \ldots, N-1, \\
&\qquad\quad x_{\mathrm{lb}} \leq x_k \leq x_{\mathrm{ub}} \qquad k = 0, \ldots, N, \\
&\qquad\quad u_{\mathrm{lb}} \leq u_k \leq u_{\mathrm{ub}} \qquad k = 0, \ldots, N-1,
\end{aligned}
\tag{16}
$$

where $\bar{x}_0$ is the given initial state, $x_{\mathrm{ub}} = (2, 4, \frac{\pi}{3}, 2), x_{\mathrm{lb}} = -x_{\mathrm{ub}}$ and $u_{\mathrm{ub}} = 25, u_{\mathrm{lb}} = -u_{\mathrm{ub}}$. The weight matrices in the cost function are $S = \mathrm{diag}(0.25, 0.025, 0.25, 0.025)$, $R = 0.0025$ and P corresponds to the solution of the discrete algebraic Riccati equation for the system linearized at $\bar{x} = (0, 0, 0, 0), \bar{u} = 0$. In order to guarantee feasibility during training the box constraints on (x, u) are softened via slack variables which are penalized in the cost with the weights $Z = \Delta t \cdot (50, 5, 50, 5, 500), z = \Delta t \cdot (0.5, 0.05, 0.5, 0.05, 5000)$ for the path constraints and $Z_e = (50, 5, 50, 5), z_e = (0.5, 0.05, 0.5, 0.05)$ for the terminal ones.

By modifying the OCP formulation (16) according to (6) and (11) we obtain the Q-function OCP and the approximate Q-function OCP, respectively.

The formulation and solution of the OCP is carried out in `acados` [24] via its Python interface. In order to speed up interactions with the OCP solver, we have used the compiled

Cython interface for the solver objects[1]. Every computation runs exclusively on one CPU thread, on a Linux Ubuntu 20.04 server with Intel Xeon E5-2687W @3.1 GHz, 16 cores and 32 GB RAM.

A. Training Setup

For approximating the MPC policy we use feed-forward Neural Networks with ReLU activation functions for the hidden layers. Additionally, after the last linear layer, we apply a $\tanh$ activation function, which bounds the output of the policy such that we fulfill the box constraints for the controls of the cart-pole OCP formulation (16). We optimize the Neural Networks by doing mini-batch stochastic gradient descent with the Adam optimizer [25].

The next important decision is on what states we want to imitate, or more specifically, on which state distribution $\mathcal{D}$ we want to minimize our imitation loss. For this, we first sample an initial state uniformly from $\mathcal{U}(\alpha \cdot \underline{x}, \alpha \cdot \bar{x})$ with $\alpha = 0.3$. We sample and discard until the drawn initial states generate optimal open-loop trajectories without constraint relaxation. Starting from the initial state, we then do a rollout with the Dagger algorithm as described in [19]. Instead of only following the expert MPC policy during the rollout, we use a mixture policy that randomly applies either the control of the expert MPC policy or our currently learned policy $\pi(x; w)$. This is done in order to generate samples that better match the distribution that we will encounter when applying the final learned policy. In more detail, we train the Neural Network for 2000 updates and every 20 updates collect additional samples by doing a rollout for 50 steps. The mixture coefficient between the expert policy and the learned policy is linearly scaled down over the training time from 1, we only use the expert, to 0, we only use the learned policy.

B. Evaluation

For evaluation, we train each algorithm for 10 different random seeds. For the initial states we either sampled from a uniform distribution with easier initial states $\alpha = 0.2$, the same distribution as during training $\alpha = 0.3$ or harder initial states with $\alpha = 0.4$. We do this to see how the performance on easier and harder initial states differ and also test the generalization capabilities of the different losses. We generate one fixed test dataset of 2000 initial states for all algorithms and seeds, with the same sampling procedure as during training. For each initial state, we then do a rollout of 50 steps for each learned policy.

We evaluate the performance of the final policies with two metrics: (1) The average rollout cost: We sum up all stage costs and slack variable costs of one rollout and then average over all rollouts and random seeds. (2) The average violation ratio, which is the ratio of rollouts that violated a constraint over all rollouts, averaged over all random seeds. Additionally, we only consider the 0.99 or 0.9 quantile to robustify our estimates against outliers, when the policy fails to stabilize the system.

[1] https://github.com/aghezz1/learning-nmpc-q-loss

TABLE I
LOSS PERFORMANCE COMPARISON

Loss	α	Quantile	Avg. Cost	Violation ratio
$\mathcal{L}^2$	0.2	0.99	0.866 ± 0.013	0.054 ± 0.042
	0.3	0.99	2.267 ± 0.250	0.225 ± 0.064
	0.4	0.90	3.229 ± 0.385	0.299 ± 0.064
$\mathcal{L}^{\mathrm{Q}}$	0.2	0.99	0.919 ± 0.093	0.005 ± 0.012
	0.3	0.99	2.308 ± 0.209	0.090 ± 0.049
	0.4	0.90	3.267 ± 0.365	0.147 ± 0.059
$\mathcal{L}^{\mathrm{Q_a}}$	0.2	0.99	0.884 ± 0.019	0.002 ± 0.001
	0.3	0.99	2.179 ± 0.081	0.088 ± 0.024
	0.4	0.90	3.076 ± 0.150	0.154 ± 0.025
$\pi^\star$ MPC	0.2	0.99	0.795 ± 0.833	–
	0.3	0.99	1.888 ± 2.027	–
	0.4	0.90	2.604 ± 2.373	–

C. Hyperparameters

To allow for a better comparison, we do a hyperparameter search in form of a grid search over the network depth $\{1, 2, 3\}$, the network width $\{64, 128, 256\}$ and the learning rate $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$. We used a fixed batch size of 32. We use the same hyperparameters for the Gauss-Newton Q-loss as for the exact Q-loss. The search objective is the average rollout cost over the same initial state dataset as described in the previous section, but only using 3 random seeds.

D. Results

In the following, we present the experimental results. In Table I, we compare the performance of the different algorithms. For the average rollout cost, we see that the exact Q-loss performs slightly worse than BC, whereas the Gauss-Newton Q-loss performs significantly better for the harder initial state dataset $\alpha = 0.4$. However, for the harder examples the gap to the original MPC policy is also significantly larger. One explanation is that for harder examples the non smoothness of the optimal control policy increases, making it harder for the network to approximate the MPC, especially for BC.

For the average violation ratio, we see that the exact Q-loss and the Gauss-Newton Q-loss perform significantly better, than BC with $\mathcal{L}^2$. This can be attributed to the fact that the exact Q-loss and the Gauss-Newton Q-loss contain constraint satisfaction and do not rely on a surrogate loss.

In Figure 3, we compare the controls of the policies learned with the different algorithms on exemplary rollouts of a trained Neural Network for one seed. We see that the original MPC shows a very non smooth control signal, which the BC loss $\mathcal{L}^2$ tries to imitate. The policies corresponding to the exact Q-loss and the Gauss-Newton Q-loss show a smoother control signal, that deviates more from the original MPC. This can be explained by the fact that the proposed losses directly optimize (approximate) versions of the OCP, thus finding their own trade-off between feasibility and optimality.

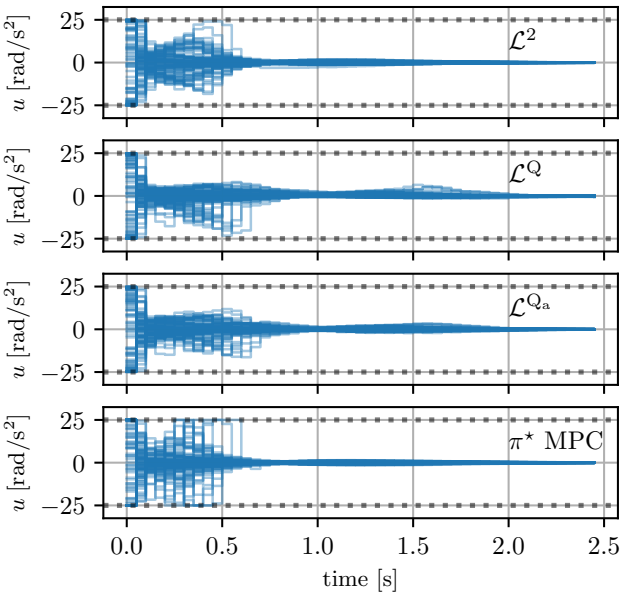

Fig. 3. Representative 100 rollouts for $\alpha = 0.3$ for one network.

TABLE II
AVERAGE GRADIENT COMPUTATION SPEED

Loss	$\mathcal{L}^2$	$\mathcal{L}^{\mathrm{Q}}$	$\mathcal{L}^{\mathrm{Q_a}}$
Speed (batch/second)	151.78	4.12	15.23

In Table II, we compare the average speed for computing the gradient on a batch (with batch size 32) for 300 iterations.

V. Conclusions

In this paper, we have presented a new loss for Imitation Learning from MPC based on the underlying OCP. This loss allows the learned policy to directly minimize the OCP performance objectives and constraint satisfaction. Compared to standard losses, the Q-loss evaluation requires the solution of a possibly nonlinear and nonconvex optimization problem for each new sample, resulting in demanding computational effort. We suggest to mitigate this issue by a second Q-loss based on the Gauss-Newton approximation of the associated OCP, therefore its evaluation requires the solution of a convex quadratic program. Finally, we have compared the policy learned using the Q-losses against Behavioral Cloning, on the control of a constrained nonlinear system. On this example, the Q-loss-based policies achieve significantly lower constraint violations and comparable closed-loop costs. In the future, we aim to test the losses on more complex examples and combine them with Reinforcement Learning.

References

[1] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 2nd ed. Nob Hill, 2017.

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

[3] A. Bemporad, F. Borrelli, and M. Morari, "The explicit solution of constrained LP-based receding horizon control," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, Sydney, Australia, 1999.

[4] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, pp. 3–20, 2002.

[5] T. A. Johansen, "Approximate explicit receding horizon control of constrained nonlinear systems," *Automatica*, vol. 40, no. 2, pp. 293–300, 2004.

[6] 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.

[7] B. M. Åkesson and H. T. Toivonen, "A neural network model predictive controller," *Journal of Process Control*, vol. 16, no. 9, pp. 937–946, 2006.

[8] S. Lucia, D. Navarro, B. Karg, H. Sarnago, and O. Lucia, "Deep learning-based model predictive control for resonant power converters," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 1, pp. 409–420, 2020.

[9] J. Drgoňa, A. Tuor, E. Skomski, S. Vasisht, and D. Vrabie, "Deep learning explicit differentiable predictive control laws for buildings," *IFAC-PapersOnLine*, vol. 54, no. 6, pp. 14–19, 2021.

[10] J. Nubert, J. Köhler, V. Berenz, F. Allgöwer, and S. Trimpe, "Safe and fast tracking on a robot manipulator: Robust mpc and neural network control," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3050–3057, 2020.

[11] R. Schwan, C. N. Jones, and D. Kuhn, "Stability verification of neural network controllers using mixed-integer programming," *IEEE Transactions on Automatic Control*, 2023.

[12] M. Hertneck, J. Köhler, S. Trimpe, and F. Allgöwer, "Learning an approximate model predictive controller with guarantees," *IEEE Control Systems Letters*, vol. 2, no. 3, pp. 543–548, 2018.

[13] R. K. Cosner, Y. Yue, and A. D. Ames, "End-to-end imitation learning with safety guarantees using control barrier functions," in *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE, 2022, pp. 5316–5322.

[14] I. Mordatch and E. Todorov, "Combining the benefits of function approximation and trajectory optimization." in *Robotics: Science and Systems*, vol. 4, 2014, p. 23.

[15] J. Carius, F. Farshidian, and M. Hutter, "Mpc-net: A first principles guided policy search," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2897–2904, 2020.

[16] A. Reske, J. Carius, Y. Ma, F. Farshidian, and M. Hutter, "Imitation learning from mpc for quadrupedal multi-gait control," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 5014–5020.

[17] R. H. Byrd, J. Nocedal, and R. A. Waltz, "Steering exact penalty methods for nonlinear programming," *Optimization Methods and Software*, vol. 23, no. 2, pp. 197–213, 2008.

[18] P. J. Huber, "Robust estimation of a location parameter," *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, 1964.

[19] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[20] D. Bertsekas, *Dynamic programming and optimal control: Volume I*. Athena scientific, 2012, vol. 1, 3rd edition.

[21] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 834–846, 1983.

[22] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*. Pmlr, 2014, pp. 387–395.

[23] R. Verschueren, N. van Duijkeren, R. Quirynen, and M. Diehl, "Exploiting convexity in direct optimal control: a sequential convex quadratic programming method," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 2016.

[24] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, Oct 2021.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.