MPC-Net: A First Principles Guided Policy Search

Jan Carius[1], Farbod Farshidian, and Marco Hutter

***Abstract*—We present an Imitation Learning approach for the control of dynamical systems with a known model. Our policy search method is guided by solutions from Model Predictive Control (MPC). Typical policy search methods of this kind minimize a distance metric between the guiding demonstrations and the learned policy. Our loss function, however, corresponds to the minimization of the control Hamiltonian, which derives from the principle of optimality. Therefore, our algorithm directly attempts to solve the optimality conditions with a parameterized class of control laws. Additionally, the proposed loss function explicitly encodes the constraints of the optimal control problem and we provide numerical evidence that its minimization achieves improved constraint satisfaction. We train a mixture-of-expert neural network architecture for controlling a quadrupedal robot and show that this policy structure is well suited for such multimodal systems. The learned policy can successfully stabilize different gaits on the real walking robot from less than 10 min of demonstration data.**

***Index Terms*—Learning from Demonstration, Legged Robots, Optimization and Optimal Control**

I. Introduction

The control of robotic systems with fast and unstable dynamics requires carefully designed feedback controllers. Hybrid, underactuated walking robots pose an especially challenging setting in this respect.

Recent successes in Reinforcement Learning (RL) demonstrate sophisticated walking robot control [1]–[5], yet a large number of policy rollouts need to be collected to reach the required performance level. It is, therefore, common practice to use physics simulators during training and subsequently attempt a sim-to-real transfer [1], [4].

Imitation Learning (IL) [6] appears to be a promising method that could reduce the sampling complexity of learning-based approaches by guiding them with expert demonstrations. When good demonstrations are available, sampling efficiency can be drastically improved over classical RL [7].

An appealing way to automatically generate such demonstrations for known dynamical systems are model-based methods such as Optimal Control (OC) and Model Predictive Control (MPC). They provide a formal framework for generating control commands that respect physical constraints and optimize a performance criterion. Knowledge of a system model and its gradients enable such methods to discover complex robot behaviors in a very sample-efficient way [8]–[13]. Unfortunately, when deploying on a robot, the entire optimization problem has to be solved online because the resulting control policy is only valid around the current state. Moreover, the robustness against disturbances – both of intrinsic nature (e.g., modeling errors) as well as external effects – is critically dependent on the assumption that a new motion plan can be generated sufficiently fast. Even for moderately complex systems, the update frequency of MPC becomes a limiting factor when deploying on onboard computers.

Learning from OC solutions has proven a viable option for robot control that combines the advantages of both approaches [14]–[21]. The benefit of using a solver as expert demonstrator over humans or animals is that there is no domain adaptation problem, and one can query demonstrations from arbitrary states. Additionally, one may request the solver to explicitly handle constraints instead of only presuming that demonstrations are constraint-consistent.

Several methods take an inverse OC approach to IL: Multiple local approximations of the value function, computed by OC runs, are aggregated into a single global approximation [22]–[24]. The learned value function and its induced optimal policy are in turn used to reduce the OC time horizon or speed up convergence. Alternatively, a Behavioral Cloning (BC) approach to IL attempts to directly learn a policy that reproduces the expert's demonstrations without maintaining a value function explicitly. Accordingly, the original RL problem is transformed into a supervised learning problem since the demonstrator's actions can be interpreted as labels.

Our proposed algorithm belongs to the family of such actor-only approaches: We introduce MPC-Net, a *policy search* method that is *guided* by an MPC algorithm to find a parametrized control policy. The method can be seen as a policy iteration scheme that draws data from a perfect critic (i.e., the MPC). Our key innovation is a theoretically motivated loss function, which is based on *first principles* from OC, namely the minimization of the control Hamiltonian. The structure of the control Hamiltonian captures the system dynamics and constraints of the control problem. We show that this learning objective has favorable properties in terms of convergence and constraint satisfaction, which is particularly important for systems interacting with the environment.

Closely related to our algorithm are policy search methods with a teacher-learner setup [17]–[19]. These works employ an OC solver as a teacher from which a policy is learned. Contrary to our work, however, the teacher adapts to the student. This assimilation is achieved by adding a penalty term to the OC cost function so that demonstrations are created that

Manuscript received: September 09, 2019; Revised December 14, 2019; Accepted January 22, 2020.

This paper was recommended for publication by Editor Dongheui Lee upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by Intel Labs, the Swiss National Science Foundation (SNF) through project 166232, 188596, the National Centre of Competence in Research Robotics (NCCR Robotics), and the European Union's Horizon 2020 research and innovation program under grant agreement No 780883. This work was conducted as part of ANYmal Research, a community to advance legged robotics.

All authors are with the Robotic Systems Lab, ETH Zürich, Switzerland. [1]jcarius@ethz.ch

Digital Object Identifier (DOI): see top of this page.

remain close to the student's policy. Additionally, the student's objective is usually the optimization of a distance metric between student's and teacher's policy outputs. However, minimizing a distance measure may not directly correspond to improved performance, e.g., in constrained settings it is usually more important to satisfy constraints rather than mimicking the teacher accurately. In our approach, no such choice of a distance metric has to be made. Notably, our learner is never presented with the optimal control input. Additionally, since our demonstrator does not adapt to the current policy of the learner, all demonstration samples remain valid and can be re-used, thereby boosting sampling efficiency.

Imitating a demonstrator that is not adaptive to the learner induces the problem of distribution matching: Inevitable approximation errors between the learned and demonstrated policies make rollouts of the learned policy encounter a different distribution of states than the one from demonstration data. Ross et al. [25], [26] show that the resulting errors can compound quadratically in the time horizon. We use elements of their proposed solutions (i.e., probabilistic mixing and dataset augmentation) to ensure that the distributions match. Simply put, we bias the demonstrator's query states towards the observations that our policy sees and thereby receive samples that match the learner's distribution better.

While the idea of policy search through minimization of the control Hamiltonian applies to arbitrary parameterized policies such as neural networks, weighted motion primitives, or spline coefficients, we consider the very general class of mixture-of-expert neural networks policies [27] in this work. Our choice caters for the fact that OC is an inverse problem with potentially multiple solutions for the same observation. The expert data may, therefore, exhibit such multimodal behavior. We show that this choice of network structure has favorable properties in terms of convergence and constraint satisfaction and is particularly suitable for controlling legged robots since these systems inherently exhibit multi-modal dynamics.

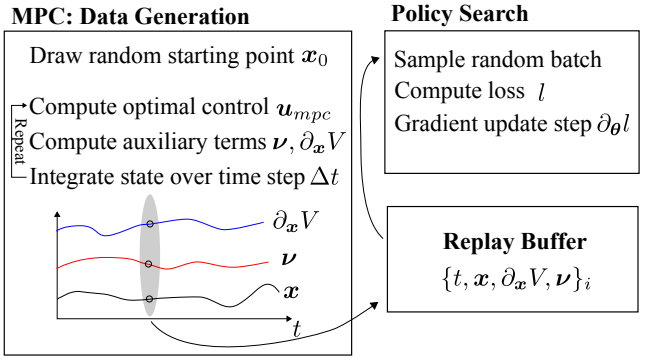

Fig. 1. Schematic of the MPC-Net policy learning approach

Algorithm 1 MPC-Net Guided Policy Learning

Given: Replay Buffer $\boldsymbol{M}$, mpcSolver
Given hyperparameters: maxIter, mpcDecimation, batchSize, learningRate, rolloutLength
for iter in [1 : maxIter] **do**
  if modulo(iter, mpcDecimation) **then**
    $\alpha \leftarrow 1-$ iter / maxIter
    $\boldsymbol{x}_0 \leftarrow$ sampleRandomStartingState
    for t_i in [0, rolloutLength] **do**
      $\boldsymbol{u}_{mpc}, \boldsymbol{K} \leftarrow$ mpcSolver(t_i, $\boldsymbol{x}_0$)
      $\boldsymbol{x} \leftarrow$ sampleInNeighborhood($\boldsymbol{x}_0$)
      $\partial_{\boldsymbol{x}} V \leftarrow$ valueFunctionDerivative($t_i, \boldsymbol{x}$)
      $\boldsymbol{\nu} \leftarrow$ constraintLagrangian($t_i, \boldsymbol{x}$)
      Append sample $\{t_i, \boldsymbol{x}, \partial_{\boldsymbol{x}} V, \boldsymbol{\nu}\}$ to $\boldsymbol{M}$
      $\boldsymbol{x}_0 \leftarrow$ stepSystem($\boldsymbol{x}_0, \alpha \boldsymbol{\pi}_{\text{mpc}} + (1-\alpha)\boldsymbol{\pi}(\boldsymbol{\theta}_{\text{iter}}, \boldsymbol{x}_0)$)
    end for
  end if
  $\boldsymbol{S} \leftarrow$ drawRandomSampleBatch($\boldsymbol{M}$, batchSize)
  $\boldsymbol{U} \leftarrow$ evaluatePolicyOnSamples($\boldsymbol{\pi}(\boldsymbol{\theta}_{\text{iter}}), \boldsymbol{S}$)
  $l \leftarrow$ computeLoss($\boldsymbol{U}, \boldsymbol{S}$)
  $\boldsymbol{\theta}_{\text{iter}+1} \leftarrow$ stepOptimizer($\partial_{\boldsymbol{\theta}} l$)
end for

Statement of Contributions

The contributions of this work are as follows:

- Derivation of a novel loss function for policy search that is based on fundamental concepts from OC
- Experiments showing that the explicit enconding of constraints in our loss function achieves improved constraint satisfaction compared to standard behavioral cloning
- Demonstration of improved efficiency in terms of MPC calls by exploiting a local approximation of the value function
- Results showing that a mixture-of-expert network architecture outperforms a general Multilayer Perceptron (MLP) for control of a walking robot
- Validation of the trained control policies on robotic hardware. The learned controllers successfully stabilize two different gaits on a quadrupedal robot

II. Method

The key steps of our method are listed in Alg. 1[1] and are schematically shown in Fig. 1. Data is generated by running MPC from a feasible, random initial state. Samples from the resulting optimal trajectories are stored in a replay buffer. At each policy update step, we construct a loss function by drawing a batch of the stored samples and perform a stochastic gradient descent step in the policy parameter space. Every mpcDecimation-th iteration, MPC produces a new set of samples that augment the dataset.
In this section, we first explain the control problem and the structure of its solution. Subsequently, we present the theoretical properties of the optimal solution and how they motivate our loss function. Finally, we show how a neural network policy is trained from demonstrations of optimal trajectories.

[1]Our implementation is openly available at `https://github.com/leggedrobotics/MPC-Net`

A. Model Predictive Control

We consider a continuous-time, finite horizon OC problem

$$\begin{aligned} \underset{\boldsymbol{u}(\cdot)}{\text{minimize}} \quad & \Phi(\boldsymbol{x}(t_f)) + \int_0^{t_f} l(\boldsymbol{x},\boldsymbol{u},t)\,\mathrm{d}t, && (1)\\ \text{subject to} \quad & \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x},\boldsymbol{u},t), \quad \boldsymbol{x}(0) = \boldsymbol{x}_0, \\ & \boldsymbol{g}(\boldsymbol{x},\boldsymbol{u},t) = \boldsymbol{0}, \\ & \boldsymbol{h}(\boldsymbol{x},\boldsymbol{u},t) \geq \boldsymbol{0}, && (2) \end{aligned}$$

where t_f is the time horizon, $\boldsymbol{x}_0$ a given initial state, $\Phi(\cdot)$ the final cost and $l(\cdot)$ the intermediate cost function. $\boldsymbol{f}(\cdot)$, $\boldsymbol{g}(\cdot)$, and $\boldsymbol{h}(\cdot)$ are time-dependent vector fields defining the system dynamics, the equality constraints, and the inequality constraints, respectively. The problem's associated optimal value function V (cost-to-go) is defined as

$$V(t,\boldsymbol{x}) = \min_{\substack{\boldsymbol{u}(\cdot),\\ \text{s.t. (2)}}} \quad \Phi(\boldsymbol{x}(t_f)) + \int_t^{t_f} l(\boldsymbol{x}(t),\boldsymbol{u}(t),t)\,\mathrm{d}t\,. \quad (3)$$

In principle, our method works with any optimization algorithm that can handle the constraints (2) and that provides an approximation of the optimal value function (3). The entire solving procedure is denoted mpcSolver in Alg. 1. In this work, we employ a variant of the Differential Dynamic Programming (DDP) algorithm called Sequential Linear-Quadratic (SLQ) control [28], which is the continuous-time equivalent to the Iterative Linear-Quadratic Regulator (iLQR) [29]. This solver handles the inequality constraints $\boldsymbol{h}(\cdot)$ through a barrier function $b(\cdot)$ [30] and explicitly computes optimal Lagrange multipliers $\boldsymbol{\nu}(\cdot)$ for satisfaction of the state-input equality constraint $\boldsymbol{g}(\cdot)$ [28]. The Lagrangian of the OC problem (1) is therefore given by

$$\begin{aligned} \mathcal{L}(\boldsymbol{x},\boldsymbol{u},t) := l(\boldsymbol{x},\boldsymbol{u},t) + \sum_i b\big(\boldsymbol{h}_i(\boldsymbol{x},\boldsymbol{u},t)\big) \\ + \boldsymbol{\nu}^\top(t,\boldsymbol{x})\boldsymbol{g}(\boldsymbol{x},\boldsymbol{u},t). \quad (4) \end{aligned}$$

The solution of problem (1) consists of nominal state and input trajectories $\{\boldsymbol{x}_{\text{nom}}(\cdot), \boldsymbol{u}_{\text{nom}}(\cdot)\}$ as well as time-dependent linear feedback gains $\boldsymbol{K}(t)$ that define the optimal control policy

$$\boldsymbol{\pi}_{\text{mpc}}(t,\boldsymbol{x}) = \boldsymbol{u}_{\text{nom}}(t) + \boldsymbol{K}(t)\left(\boldsymbol{x} - \boldsymbol{x}_{\text{nom}}(t)\right). \quad (5)$$

As a byproduct of the solver, we also have access to the state derivative of the value function $\partial_{\boldsymbol{x}} V$.

During our emulated real-time MPC loop, we let the solver compute the optimal policy, then store the values of $\{t, \boldsymbol{x}, \partial_{\boldsymbol{x}} V, \boldsymbol{\nu}\}$ at the first time step of the solution in our replay memory. Next, we update the current state using the system dynamics and continue until the rollout length is reached.

B. Policy Loss Function

It is a known property of OC [31, pp. 111–120] that the optimal input $\boldsymbol{u}^*(t)$ must satisfy

$$\begin{aligned} \boldsymbol{u}^*(t,\boldsymbol{x}) &= \arg\min_{\boldsymbol{u}} \mathcal{H}(\boldsymbol{x},\boldsymbol{u},t)\,, && (6)\\ \mathcal{H}(\boldsymbol{x},\boldsymbol{u},t) &:= \mathcal{L}(\boldsymbol{x},\boldsymbol{u},t) + \partial_{\boldsymbol{x}} V(t,\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x},\boldsymbol{u},t)\,, && (7) \end{aligned}$$

where $\mathcal{H}(\cdot)$ is the control Hamiltonian, which directly arises from the Hamilton–Jacobi–Bellman (HJB) equation. Moreover, under some sufficient conditions (so-called Weierstrass conditions), the Hamiltonian attains a strong minimum at $\boldsymbol{u}^*$. Its minimization can, therefore, be seen as a recipe for finding the optimal controls.

A globally optimal policy would have to satisfy (6) at *any* $t, \boldsymbol{x}$. This minimization, however, has a great drawback, which is commonly referred to as the "curse of dimensionality". Furthermore, recording the solution of every time-state pair requires an enormous amount of storage, which is impractical even for moderate-size systems.

We turn to function approximation as a remedy for these difficulties and introduce a parameterized policy $\boldsymbol{\pi}(t,\boldsymbol{x}|\boldsymbol{\theta})$. Our problem is now to find some parameters $\boldsymbol{\theta}^*$ such that $\boldsymbol{\pi}(\cdot)$ maps a given $(t,\boldsymbol{x})$ pair to a control that achieves a minimum of $\mathcal{H}$. The substitution of the pointwise optimal $\boldsymbol{u}^*$ with the parameterized policy may introduce an optimality gap between the minima $\mathcal{H}(\boldsymbol{x},\boldsymbol{u}^*,t)$ and $\mathcal{H}(\boldsymbol{x},\boldsymbol{\pi}(\boldsymbol{x},t|\boldsymbol{\theta}^*),t)$. We can relate the size of this optimality gap to the discrepancy in optimal controls through the following lemma:

Lemma 1. *Given that $\boldsymbol{u}^*$ is a strong minimum of equation* (6) *which satisfies the Weierstrass sufficient condition of optimality. Then the optimality gap in the pointwise minimization of the control Hamiltonian* (6) *upper bounds the distance to the optimal control according to*

$$||\boldsymbol{\pi} - \boldsymbol{u}^*||^2 \leq \frac{2}{\delta}(\mathcal{H}(\boldsymbol{x},\boldsymbol{\pi},t) - \mathcal{H}(\boldsymbol{x},\boldsymbol{u}^*,t))\,, \quad (8)$$

where $\delta > 0$ pertains to the smallest eigenvalue of $\partial^2_{\boldsymbol{u}}\mathcal{H}$ in the neighborhood of the optimal control, $\boldsymbol{u}^*$.

Proof. The proof is provided in Appendix V. □

This statement implies that $\boldsymbol{\pi}(t,\boldsymbol{x}|\boldsymbol{\theta}^*)$ is approaching the optimal control for a specific $(t,\boldsymbol{x})$ pair as the optimality gap in $\mathcal{H}$ is reduced. It is unrealistic to minimize this gap for every point in state-space simultaneously because that would require an extremely flexible parametrization of $\boldsymbol{\pi}$ and would also assume that $\mathcal{H}$ is known at every state.

To our benefit, however, SLQ computes the value function along trajectories in state space. These trajectories induce a distribution over time and state $\{t,\boldsymbol{x}\} \sim \mathcal{P}$ that encodes which areas of the time-state-space are visited by an optimal controller. For our purposes, it is, therefore, sufficient to minimize the optimality gap almost everywhere with respect to $\mathcal{P}$. Taking the expectation of (8) gives

$$\mathbb{E}_{\mathcal{P}}\left[||\boldsymbol{\pi} - \boldsymbol{u}^*||^2\right] \leq \frac{2}{\delta}\,\mathbb{E}_{\mathcal{P}}\left[\mathcal{H}(\boldsymbol{x},\boldsymbol{\pi},t) - \mathcal{H}(\boldsymbol{x},\boldsymbol{u}^*,t)\right], \quad (9)$$

Restricting the minimization only to the relevant states (i.e., those with nonzero probability mass) also allows us to invoke the Universal Approximation Theorem for our parameterized policy: We can expect to find a $\boldsymbol{\theta}^*$ that makes the expected optimality gap in $\mathcal{H}$ arbitrarily small if the function class $\boldsymbol{\pi}(t,\boldsymbol{x}|\boldsymbol{\theta})$ is sufficiently rich. Our strategy for finding the optimal parameters is, therefore, given by

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}}\ \mathbb{E}_{\{t,\boldsymbol{x}\}\sim\mathcal{P}}\left[\mathcal{H}(\boldsymbol{x},\boldsymbol{\pi}(t,\boldsymbol{x}|\boldsymbol{\theta}),t)\right]. \quad (10)$$

The quantity in the expectation (10) can be seen as a per-sample loss for policy training. It is essential to realize that the control Hamiltonian allows us to find the optimal control via this unconstrained minimization because the future cost and constraint Lagrangian have already been included. It is, therefore, not necessary to perform Monte-Carlo-style rollouts to find the optimal control.

The MPC loop presented in Sec. II-A serves as a data generation mechanism for the policy search module. In general terms, the MPC fills a replay buffer with data points that correspond to the states that it has encountered, and those tuples are sampled from to compute the empirical expectation in (10). In our implementation, the samples for computing the policy gradient are drawn uniformly at random from the replay buffer in order to break their temporal correlation [32].

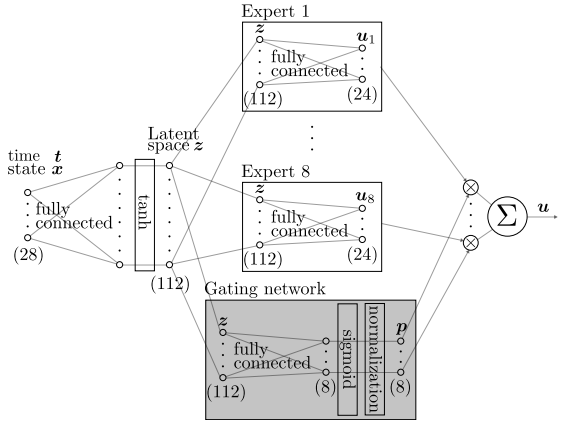

Fig. 2. Architecture of our mixture-of-experts network. The dimensions correspond to the instantiation for the ANYmal robot.

C. Augmenting Samples Using the Optimal Solution

A favorable property of SLQ, being a local dynamic programming approach, is that it computes a second-order approximation of the optimal value function, as well as a first-order approximation of the Lagrange multiplies and control policy, in the vicinity of the optimal state-input trajectories. In turn, the control Hamiltonian can also be approximated in a region around the optimal solution. Our numerical investigation in Sec. III-B demonstrates that this approximated Hamiltonian still yields an acceptable estimation of the optimal control input for samples neighboring the MPC trajectories. This observation motivates us to extract samples not only from the MPC generated trajectories but also from a tube surrounding these trajectories, which further improves the sample complexity of the approach.

By initializing the the control problem (1), (2) at feasible, random starting points, the areas where the value function is known corresponds to the subset of states that are visited by a (close-to) optimal policy. This fact can be exploited to increase the extracted informational content from a rollout of the optimal policy. By sampling around the nominal state, our data automatically covers tubes in state space, which accelerates learning and makes the learned policy more robust. This procedure, denoted sampleInNeighborhood in Alg. 1, amounts to drawing states from a Gaussian distribution according to

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}_{\text{nom}}, \Sigma_{\boldsymbol{x}}), \tag{11}$$

where the covariance matrix has diagonal entries corresponding to the typical disturbance that the respective state component may encounter. The sampling idea is conceptually similar to fitting the tangent space of the demonstrator policy instead of just the nominal control command [16].

D. Addressing Distribution Mismatch

Unfortunately, despite our efforts to extract samples from trajectories that cover a large volume in state space, there is still a bias of the state distribution towards those states that are encountered by the optimal policy. This distribution mismatch is a common problem in IL and stems from the fact that a learned controller produces inevitably different control inputs than the demonstrator (even when fully converged, unstable physical systems may amplify small differences), which will eventually drive the system into an area of the state space from which no data is available. To avoid this scenario, we use a behavioral policy $\boldsymbol{\pi}_{\mathcal{B}}(\cdot)$ to push the emulated MPC loop towards the states that will be seen by the learned policy. Taking inspiration from Dagger [26], the update rule for the next state (stepSystem method in Alg. 1) is given by

$$\boldsymbol{x}(t + \Delta t) = \boldsymbol{x}(t) + \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{\pi}_{\mathcal{B}}(t, \boldsymbol{x}, \boldsymbol{\theta}))\Delta t, \tag{12}$$

$$\boldsymbol{\pi}_{\mathcal{B}}(t, \boldsymbol{x}, \boldsymbol{\theta}) = (1 - \alpha)\boldsymbol{\pi}_{\text{mpc}} + \alpha\boldsymbol{\pi}(t, \boldsymbol{x}|\boldsymbol{\theta}), \tag{13}$$

where the mixing parameter α is initially zero and linearly increases with the number of iterations until it has reached one in the final iteration. Through this process, the learned policy is gradually given more responsibility to decide where the MPC algorithm should be applied. It is important to note that the MPC solver is not influenced by the learned policy and produces optimal solutions independent of the value of α.

E. Policy Structure and Training

Now that the loss function and a way to populate our experience buffer is defined, we turn the actual training procedure and computation of stochastic gradients of our policy.

In this work, we use a mixture-of-experts architecture [27] for the control policy, shown in Fig. 2. Allowing multiple policies $\boldsymbol{\pi}_i$ to compete naturally handles the non-uniqueness of the OC solution. For example, passing an obstacle around the left or right side may be an equally good choice that two different experts will try to imitate, but forcing a monolithic network to interpolate between these solutions can be catastrophic.

The final control output of the network is a convex combination of the outputs of different expert sub-policies

$$\boldsymbol{\pi}(t, \boldsymbol{x}|\boldsymbol{\theta}) = \sum_{i=1}^{\text{NumExperts}} p_i(t, \boldsymbol{x}|\boldsymbol{\theta})\, \boldsymbol{\pi}_i(t, \boldsymbol{x}|\boldsymbol{\theta})\,. \tag{14}$$

The mixing coefficients p_i are the output of a gating network whose final activation ensures that all coefficients are positive and sum up to one. While a softmax layer achieves this constraint, we find that a sigmoid activation with subsequent normalization performs better in selecting a consistent number of experts for a given task across multiple training runs. We believe the reason for this observation is that the softmax

Fig. 3. The quadrupedal robot ANYmal. The floating base and three joints per leg amount to 18 DOF. Our kinodynamic model of this robot has 24 states and 24 inputs.

activation is too sharp in selecting one specific expert such that an unlucky initialization may lead to some experts never even being considered and therefore not receiving policy updates.

Both the expert sub-policies and the gating network share a common latent space representation. The overall policy (14) remains a feed-forward neural network and can, therefore, be trained with standard deep learning optimization techniques: At each policy iteration step, we draw a batch of $\{t, \boldsymbol{x}, \partial_{\boldsymbol{x}} V, \boldsymbol{\nu}\}$ tuples from the replay buffer and compute the empirical loss for this batch as

$$\text{loss} = \sum_{j=1}^{\text{BatchSize}} \sum_{i=1}^{\text{NumExperts}} p_i(t_j, \boldsymbol{x}_j|\boldsymbol{\theta}) \mathcal{H}(\boldsymbol{x}_j, \boldsymbol{\pi}_i(t_j, \boldsymbol{x}_j|\boldsymbol{\theta}), t_j) \ . \tag{15}$$

Note that we force each experts' output to individually minimize the Hamiltonian to encourage specialization [27]. This procedure is slightly different from inserting (14) into (10), which would only encourage their combined output to be optimal. Training the optimal policy involves taking gradient steps in the parameter space. The policy gradient for the loss function (15) for a given sample j is equal to

$$\begin{aligned} &\sum_{i=1}^{N_{\text{experts}}} p_i(t_j, \boldsymbol{x}_j|\boldsymbol{\theta}) \partial_{\boldsymbol{u}} \mathcal{H}(\boldsymbol{x}_j, \boldsymbol{\pi}_i(t_j, \boldsymbol{x}_j|\boldsymbol{\theta}), t_j) \partial_{\boldsymbol{\theta}} \boldsymbol{\pi}(t_j, \boldsymbol{x}_i|\boldsymbol{\theta}) \\ &\quad + \partial_{\boldsymbol{\theta}} p_i(t_j, \boldsymbol{x}_j|\boldsymbol{\theta}) \mathcal{H}(\boldsymbol{x}_j, \boldsymbol{\pi}_i(t_j, \boldsymbol{x}_j|\boldsymbol{\theta}), t_j). \end{aligned} \tag{16}$$

For all nominal states the control derivative of the Hamiltonian $\partial_{\boldsymbol{u}} \mathcal{H}$ is computed as a byproduct of solving the problem (1), (2); for neighboring states the derivative of (7) can be queried. The gradients of p and $\boldsymbol{\pi}$ are calculated by backpropagation.

III. Results

We assess the policy structure and loss function of the MPC-Net algorithm separately to highlight the performance of our method and justify individual design choices.

A. Experimental Setup

The results presented in this document are produced with the quadrupedal robot ANYmal (Fig. 3), which is an example of a hybrid system with time-varying flow map and constraints. The constraints encode zero contact forces for a foot in swing phase and zero velocity when in stance phase. Our kinodynamic model amounts to 24 states (base pose, base twist, joint angles) and 24 control inputs (joint velocities, foot contact forces). The control commands from our policy are fed to a whole-body tracking controller that computes the final actuator torque commands. Instead of providing the absolute time to the network, it is more expedient to encode the phase of the gait cycle of the legged robot. By abuse of notation, we, therefore, define four 'time' variables, one per leg, which are zero during stance phases and describe half a period of a sine wave during the swing motion.

TABLE I
Hyperparameters of MPC-Net

maxIter	100'000	mpcDecimation	500
rolloutLength	3 s	Replay Buffer Size	100'000
time step Δt	0.0025 s	N_{experts}	8
learningRate	1e-3	batchSize	32

We use a quadratic OC cost function (1) of the form

$$\Phi(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{x}_{\text{ref},f})^\top \boldsymbol{Q}_f (\boldsymbol{x} - \boldsymbol{x}_{\text{ref},f}) \ , \tag{17}$$

$$l(\boldsymbol{x}, \boldsymbol{u}, t) = (\boldsymbol{x} - \boldsymbol{x}_{\text{ref}}(t))^\top \boldsymbol{Q} (\boldsymbol{x} - \boldsymbol{x}_{\text{ref}}(t)) + \boldsymbol{u}^\top \boldsymbol{R} \boldsymbol{u}. \tag{18}$$

The reference states encourage the system to return to the origin with a trotting or static walk gait and then maintain a nominal configuration. Our quadratic cost structure, together with the fact that our constraints and dynamics are input-affine, makes the Hamiltonian a quadratic function in $\boldsymbol{u}$. Notably, this guarantees that all assumptions made for the proof of Lemma 1 are fulfilled.

Since our loss function (15) directly depends on the sampled data, it is not a suitable termination criterion for the training process and has a high variance. We monitor the training progress of our policy by computing a rollout of the system dynamics $\boldsymbol{f}(\cdot)$ with the learned policy from random initial points. A rollout lasts 3 s but is terminated early if the pitch or roll angle exceed 30 ° or the height deviates more than 20 cm from the nominal value. This procedure can be seen as a test set for our learning approach. The resulting average rollout cost (1) and the survival time are good performance indicators for the policy.

All hyper-parameters of our algorithm are summarized in Tab. I. The network weights are randomly initialized before training and optimized with the AMSGrad variant of the Adam optimizer [33], [34], which implements the stepOptimizer primive in Alg. 1. We take the data from MPC as is without any pruning of failed rollouts or outlier states. For the following comparisons, we execute five training runs for each configuration and average the results. For better interpretability, the progression of training is shown in terms of the total duration of accumulated rollouts rather than (linearly related) optimizer iterations.

B. Loss Function

We begin by providing numerical evidence that minimizing the control Hamiltonian $\mathcal{H}$ yields optimal controls. To this end, we compare the optimal policy from MPC (5) with the result of the minimization (6). Table II shows median constraint violation and relative deviation from the optimal input for 40 randomly drawn points on or near optimal trajectories.

TABLE II
COMPARISON BETWEEN THE MPC POLICY AND HAMILTONIAN MINIMIZATION. WE SHOW MEDIAN CONSTRAINT VIOLATION $||\boldsymbol{g}||$ AND THE RELATIVE ERROR TO THE OPTIMAL CONTROL $\boldsymbol{u}^*$

	states on opt. trajectory		states near opt. trajectory	
	$\|\|\boldsymbol{g}\|\|$	$\frac{\|\|\boldsymbol{u}-\boldsymbol{u}^*\|\|}{\|\|\boldsymbol{u}\|\|}$	$\|\|\boldsymbol{g}\|\|$	$\frac{\|\|\boldsymbol{u}-\boldsymbol{u}^*\|\|}{\|\|\boldsymbol{u}\|\|}$
$\boldsymbol{\pi}_{\text{mpc}}$	3.44e-6	0.0	3.58e-4	2.48e-2
$\arg\min \mathcal{H}$	3.46e-4	1.58e-3	5.40e-4	2.80e-2

For query states $\tilde{\boldsymbol{x}}$ near the optimal trajectory the benchmark control $\boldsymbol{u}^*$ is computed by solving (1) for $\boldsymbol{x}_0 = \tilde{\boldsymbol{x}}$. The values confirm that our estimation of $\mathcal{H}$ is sufficiently accurate and that its minimization produces constraint-satisfactory control commands. Additionally, this result suggests that the requirement for Lemma 1 (i.e., the existence of a strong minimum) also holds for states in the vicinity of an optimal trajectory.

C. Comparison to Behavior Cloning

The next experiment compares our proposed Hamiltonian (10) as a loss function with a simpler BC loss that encourages matching of the demonstrator's control command

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\{t,\boldsymbol{x}\}\sim\mathcal{P}} ||\boldsymbol{\pi}_{\text{mpc}}(t,\boldsymbol{x}) - \boldsymbol{\pi}(t,\boldsymbol{x}|\boldsymbol{\theta})||_{\boldsymbol{R}} \,. \quad (19)$$

We use the control cost matrix $\boldsymbol{R}$ here to normalize the different control dimensions. We see in Fig. 4 that the simpler loss (19) results in similar convergence to a stable control law, but the Hamiltonian loss consistently achieves a lower constraint violation value. Constraint violation means that physical feasibility is violated, effectively allowing the robot to stabilize by cheating. When deployed in a physics simulator, the policy trained on (19) tends to fall after a few footsteps as violations errors accumulate because cheating is not possible anymore.

We conjecture that the structure of the Hamiltonian, which includes constraint violation penalties explicitly, encourages the learning algorithm to respect constraints more carefully than in the case of only observing constraint-consistent demonstrations. Note that our loss would inform the learner about constraint violations even if the demonstrations violated them.

D. Sample Efficiency

We show in Fig. 5 how sampling around the optimal trajectory influences the learning process for a quadruped walking motion. There is no noticeable effect in the loss function (i.e., the value of the Hamiltonian) throughout the process, which also suggests that this value is not a good indicator for the actual performance of the policy. Instead, a clear effect can be seen in the progression of the survival time. The plot suggests that the additionally sampled states provide valuable information for the training algorithm to learn faster and stabilize the system more consistently at the end of the training. More importantly even, we observe that the policy that is trained only on nominal samples is overly aggressive to small deviations in the system's state. These strong gains lead to oscillatory behavior when deployed on the real system, where sensors and the state estimator inevitably

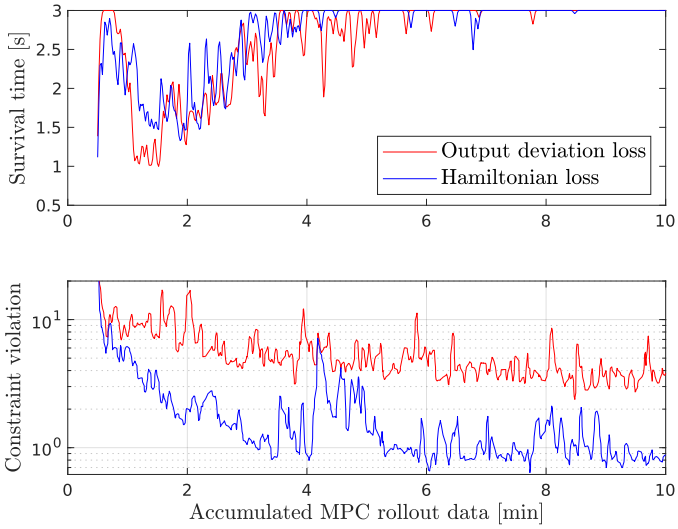

Fig. 4. Comparison between minimization of the control Hamiltonian and a simpler loss penalizing differences in policy outputs. Both loss functions are applied to the mixture-of-expert network architecture.

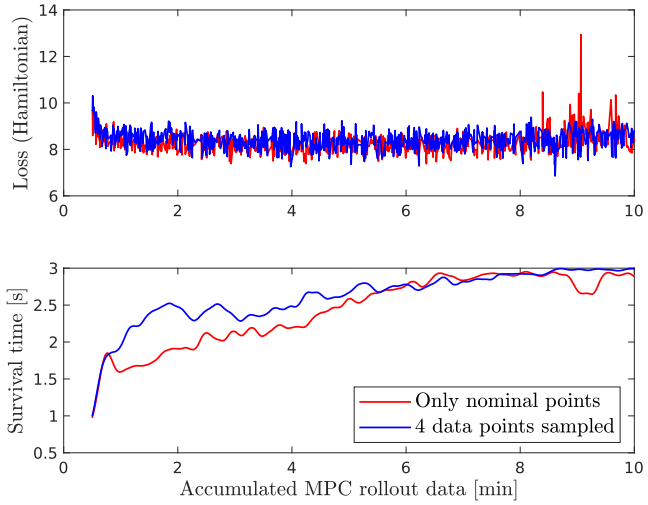

Fig. 5. Effect of collecting additional samples around the optimal trajectory. The maximum duration of a policy rollout is 3 s. Five independent experiments are averaged for each setting.

introduce noise. Subsequently, only the policy that is trained with additional samples around the optimal trajectory is robust and smooth enough to stabilize the system under noisy state estimates. Evidence of this result is shown in the video[2].

Finally, experiments show that the policies with sampling become usable on the robot at approximately 75% of the maximum number of iterations, indicating that sampling also improves the effective amount of information extracted from demonstrated trajectories and thereby necessitating fewer MPC calls. Our algorithm, therefore, learns to stabilize a walking robot from an experience buffer that is equivalent to running the robot for nine minutes with an optimal controller. Notably, this time scale opens up the possibility of learning directly on a real system.

[2]https://youtu.be/VI7wt5PCJ14

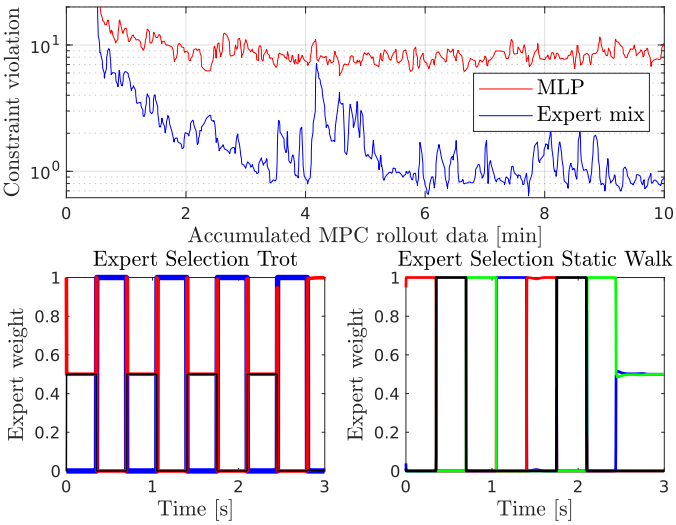

Fig. 6. The top graph shows a comparison of constraint violation during training between the expert mixture network and a MLP of equivalent size. The bottom two graphs display the output of the expert gating network for two different gaits (one color per expert). Switching times correspond exactly to changes in the contact configuration and the pattern repeats periodically with the period of the gait.

E. Mixture-of-Expert Architecture

In this experiment we compare the performance of our mixture-of-expert architecture to a classical two-layer MLP

$$\boldsymbol{\pi}_{\text{MLP}} = \boldsymbol{A}_2(\tanh(\boldsymbol{A}_1\boldsymbol{x} + \boldsymbol{b}_1)) + \boldsymbol{b}_2 \ , \tag{20}$$

with an equally-sized latent space than the one of the expert mixture.[3] While both architectures achieve similar convergence to a stable controller, Fig. 6 shows that the expert mixture reaches a significantly better constraint violation score.

We allow the expert mixture network to use 8 experts for training. Interestingly, the gating network decides to use fewer experts, and swiching between these sub-policies happens precisely at the times when the contact configuration of the system changes. For a trotting gait, only three experts are needed (blue expert for the first pair of diagonal legs, a mixture of red and black for the other pair, and red for the final stance phase) while a static walk selects four experts, one per swing leg. This result shows that the policy learns to select an appropriate expert in different domains of the state space. Moreover, a specialized expert that focuses only on a specific contact configuration learns to obey the constraints better than a single policy for all phases of the gait.

F. Robot Control

Finally, we test our trained policy on the physical ANYmal robot. The on-board policy evaluation takes approx. 0.125 ms, compared to 38 ms of an MPC update, and can therefore be called synchronously to the tracking controller. We verify that both a trotting and a static walk gait can be learned from the MPC oracle using the same network structure and identical hyperparameters. Despite the seemingly more stable static walk, both gaits pose a comparable level of difficulty to the learning algorithm which manifests in similar convergence

[3]We also tested deeper and wider MLP architectures but could not observe improved performance.

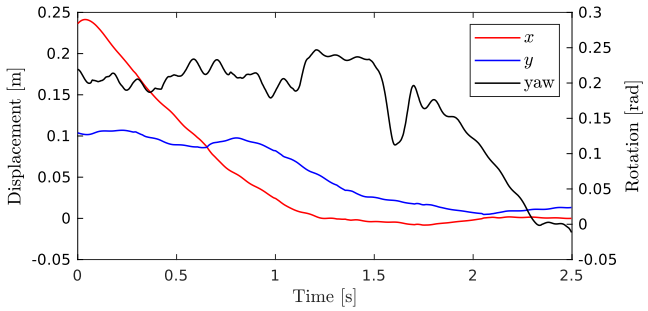

Fig. 7. Time evolution of ANYmal's base position and yaw angle under the trained policy. All quantities return to zero with minimal overshoot.

properties. The attached video shows the robot's behavior under our learned policy.

We test the policy's ability to return to the origin by starting the robot at a nonzero initial displacement and yaw rotation. In Figure 7, we plot the resulting state trajectories of x-y position as well as yaw angle, confirming that the network succeeds in the regularization task without overshoot.

IV. Conclusion

In this work, we explored a variant of MPC-guided policy search to learn a feedback control law. Contrary to other imitation learning approaches, which try to mimic the control commands of a teacher, our formulation is based on minimizing the control Hamiltonian. The optimization corresponds to solving the OC problem with a restricted family of control laws. We show that our algorithm is capable of learning a feedback policy for two different gaits of a walking robot from less than 10 minutes of demonstration data.

By design, our method cannot outperform the MPC policy because it optimizes the same cost function, and we cannot learn in areas where the optimization algorithm does not converge. However, the improved speed in control evaluation may very well stabilize motions that were not possible before or enable online control altogether. Even if the MPC algorithm is too slow to stabilize the robot, the sample efficiency of our methods facilitates learning directly on the hardware. To this end, one may compute the MPC solution online on a powerful, off-board machine.

Online MPC may also be too energy-consuming for longer autonomous operation, in which case MPC-Net could control the robot by default and MPC is only queried as soon as unknown states are encountered. Such an operating mode would ensure the safety of the system while generating more training data in regions of the state-space that are still uncertain.

A limitation related to imitating optimized trajectories is the lack of exploration, as our policy search method will fall into the same local minima that the MPC optimizer found. Future research is necessary to investigate how policies could systematically request new samples from the MPC to improve in areas where the optimal control is still uncertain.

V. Appendix: Proof to Lemma 1

Let $\boldsymbol{p} := \boldsymbol{\pi} - \boldsymbol{u}^*$ be the difference between an arbitrary policy $\boldsymbol{\pi}$ and the optimal controls $\boldsymbol{u}^*$ for a single time and state. We define $\bar{\mathcal{H}} : [0, 1] \to \mathbb{R}$ as

$$\bar{\mathcal{H}}(\alpha) = \mathcal{H}(\boldsymbol{x}, \boldsymbol{u}^* + \alpha\boldsymbol{p}, t) \ , \tag{21}$$

where α is an interpolation parameter. Assuming second order differentiability of $\mathcal{H}$, the Fundamental Theorem of Calculus allows us to write

$$\bar{\mathcal{H}}(1) - \bar{\mathcal{H}}(0) = \int_0^1 \partial_{\boldsymbol{u}} \bar{\mathcal{H}}(\alpha)\, \boldsymbol{p}\, \mathrm{d}\alpha\ . \tag{22}$$

Applying the same idea again to the integrand, we get

$$\partial_{\boldsymbol{u}} \bar{\mathcal{H}}(\alpha) - \underbrace{\partial_{\boldsymbol{u}} \bar{\mathcal{H}}(0)}_{=\mathbf{0}} = \int_0^\alpha \boldsymbol{p}^\top \partial^2_{\boldsymbol{u}} \bar{\mathcal{H}}(\beta)\, \mathrm{d}\beta\ , \tag{23}$$

with $\partial^2_{\boldsymbol{u}} \bar{\mathcal{H}}$ denoting the Hessian matrix w.r.t. $\boldsymbol{u}$. The second term above vanishes because $\boldsymbol{u}^*$ is optimal. Now we can substitute (23) into (22), resulting in

$$\bar{\mathcal{H}}(1) - \bar{\mathcal{H}}(0) = \boldsymbol{p}^\top \left(\int_0^1 \int_0^\alpha \partial^2_{\boldsymbol{u}} \bar{\mathcal{H}}(\beta)\, \mathrm{d}\beta\, \mathrm{d}\alpha \right) \boldsymbol{p}\ . \tag{24}$$

Assuming that $\boldsymbol{u}^*$ is a strong minimum which satisfies the Weierstrass sufficient condition, the Hessian of the Hamiltonian is positive definite in the neighborhood of the optimal input. Thus for small enough $\boldsymbol{p}$, there exists a positive scalar $\delta > 0$ for which

$$\partial^2_{\boldsymbol{u}} \bar{\mathcal{H}}(\beta) > \delta \boldsymbol{I},\ \text{for all}\ \beta \in [0, 1] \tag{25}$$

We then have,

$$\boldsymbol{p}^\top \left(\int_0^1 \int_0^\alpha \partial^2_{\boldsymbol{u}} \bar{\mathcal{H}}(\beta)\, \mathrm{d}\beta\, \mathrm{d}\alpha \right) \boldsymbol{p} > \frac{\delta}{2} \boldsymbol{p}^\top \boldsymbol{p}\,. \tag{26}$$

Replacing the left side of the inequality with $\bar{\mathcal{H}}(1) - \bar{\mathcal{H}}(0)$ and recalling the definition of $\boldsymbol{p}$ yields the statement of Lemma 1. □

REFERENCES

[1] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *Robotics: Science and Systems XIV*, 2018.

[2] A. Iscen, K. Caluwaerts, J. Tan, T. Zhang, E. Coumans, V. Sindhwani, and V. Vanhoucke, "Policies modulating trajectory generators," in *Conf. on Robot Learning (CoRL)*, 2018, pp. 916–926.

[3] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," in *Robotics: Science and Systems XV*, 2019.

[4] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, 2019.

[5] Z. Xie, P. Clary, J. Dao, P. Morais, J. W. Hurst, and M. van de Panne, "Iterative reinforcement learning based design of dynamic locomotion skills for cassie," *CoRR*, vol. abs/1903.09537, 2019.

[6] 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.

[7] W. Sun, A. Venkatraman, G. J. Gordon, B. Boots, and J. A. Bagnell, "Deeply aggrevated: Differentiable imitation learning for sequential prediction," in *Int. Conf. on Machine Learning ICML*, 2017, pp. 3309–3318.

[8] H. Park, P. M. Wensing, and S. Kim, "Online planning for autonomous running jumps over obstacles in high-speed quadrupeds," in *Robotics: Science and Systems XI*, 2015.

[9] M. Naveau, M. Kudruss, O. Stasse, C. Kirches, K. Mombaur, and P. Souères, "A reactive walking pattern generator based on nonlinear model predictive control," *IEEE Robotics and Automation Letters*, vol. 2, no. 1, pp. 10–17, 2017.

[10] F. Farshidian, E. Jelavic, A. Satapathy, M. Giftthaler, and J. Buchli, "Real-time motion planning of legged robots: A model predictive control approach," in *IEEE-RAS Int. Conf. on Humanoid Robotics (Humanoids)*, 2017, pp. 577–584.

[11] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1560–1567, 2018.

[12] M. Neunert, M. Stäuble, M. Giftthaler, C. D. Bellicoso, J. Carius, C. Gehring, M. Hutter, and J. Buchli, "Whole-body nonlinear model predictive control through contacts for quadrupeds," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1458–1465, 2018.

[13] J. Carius, R. Ranftl, V. Koltun, and M. Hutter, "Trajectory optimization for legged robots with slipping motions," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 3013–3020, 2019.

[14] N. D. Ratliff, D. M. Bradley, J. A. Bagnell, and J. E. Chestnutt, "Boosting structured prediction for imitation learning," in *Advances in Neural Information Processing Systems*, 2006, pp. 1153–1160.

[15] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *Int. J. Robotics Res.*, vol. 29, no. 13, pp. 1608–1639, 2010.

[16] I. Mordatch and E. Todorov, "Combining the benefits of function approximation and trajectory optimization," in *Robotics: Science and Systems X*, 2014.

[17] S. Levine and V. Koltun, "Guided policy search," in *Int. Conf. on Machine Learning ICML*, 2013, pp. 1–9.

[18] ——, "Learning complex neural network policies with trajectory optimization," in *Int. Conf. on Machine Learning ICML*, 2014, pp. 829–837.

[19] G. Kahn, T. Zhang, S. Levine, and P. Abbeel, "PLATO: policy learning using adaptive trajectory optimization," in *IEEE Int. Conf. on Robotics and Automation ICRA*, 2017, pp. 3342–3349.

[20] S. Choudhury, A. Kapoor, G. Ranade, S. Scherer, and D. Dey, "Adaptive information gathering via imitation learning," in *Robotics: Science and Systems XIII*, 2017.

[21] Y. Yang, K. Caluwaerts, A. Iscen, T. Zhang, J. Tan, and V. Sindhwani, "Data efficient reinforcement learning for legged robots," *CoRR*, vol. abs/1907.03613, 2019.

[22] C. G. Atkeson and J. Morimoto, "Nonparametric representation of policies and value functions: A trajectory-based approach," in *Advances in Neural Information Processing Systems NIPS*, 2002, pp. 1611–1618.

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

[24] N. Mansard, A. DelPrete, M. Geisert, S. Tonneau, and O. Stasse, "Using a memory of motion to efficiently warm-start a nonlinear predictive controller," in *IEEE Int. Conf. on Robotics and Automation ICRA*, 2018, pp. 2986–2993.

[25] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Int. Conf. on Artificial Intelligence and Statistics AISTATS*, 2010, pp. 661–668.

[26] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Int. Conf. on Artificial Intelligence and Statistics AISTATS*, 2011, pp. 627–635.

[27] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Computation*, vol. 3, no. 1, pp. 79–87, 1991.

[28] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An efficient optimal planning and control framework for quadrupedal locomotion," in *IEEE Int. Conf. on Robotics and Automation ICRA*, May 2017, pp. 93–100.

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

[30] R. Grandia, F. Farshidian, R. Ranftl, and M. Hutter, "Feedback MPC for torque-controlled legged robots," *CoRR*, vol. abs/1905.06144, 2019.

[31] D. P. Bertsekas, *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific, 2005.

[32] L. J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, pp. 293–321, 1992.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Int. Conf. on Learning Representations ICLR*, 2015.

[34] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," in *Int. Conf. on Learning Representations ICLR*, 2018.