

# Infinite-Horizon Value Function Approximation for Model Predictive Control

Armand Jordana<sup>1</sup>, Sébastien Kleff<sup>1,4</sup>, Arthur Haffemayer<sup>3</sup>, Joaquim Ortiz-Haro<sup>1</sup>,  
Justin Carpentier<sup>2</sup>, Nicolas Mansard<sup>3,5</sup> and Ludovic Righetti<sup>1,5</sup>

**Abstract**—Model Predictive Control has emerged as a popular tool for robots to generate complex motions. However, the real-time requirement has limited the use of hard constraints and large preview horizons, which are necessary to ensure safety and stability. In practice, practitioners have to carefully design cost functions that can imitate an infinite horizon formulation, which is tedious and often results in local minima. In this work, we study how to approximate the infinite horizon value function of constrained optimal control problems with neural networks using value iteration and trajectory optimization. Furthermore, we experimentally demonstrate how using this value function approximation as a terminal cost provides global stability to the model predictive controller. The approach is validated on two toy problems and a real-world scenario with online obstacle avoidance on an industrial manipulator where the value function is conditioned to the goal and obstacle.

## I. INTRODUCTION

Model Predictive Control (MPC) has demonstrated its ability to plan efficiently online on complex robots [1]–[4]. However, in robotics applications, MPC exhibits local behaviors for two main reasons. The first is that practitioners often rely on local Trajectory Optimization (TO) techniques. The second is the use of a finite horizon which creates local minima. Furthermore, without an appropriate design of the terminal cost and constraint set, MPC with a finite horizon is not guaranteed to be globally stable or recursively feasible [5], [6]. In practice, this has led roboticists to spend a lot of time designing cost functions to avoid local minima. Infinite horizon constrained MPC is a compelling framework as it ensures global stability [6]. In other words, it guarantees that any initial state converges to a zero-cost stationary state while satisfying hard constraints. Unfortunately, in the general case, the problem is intractable and has to be approximated [7].

Manuscript received: January 31, 2025; Revised: April 28, 2025; Accepted: May 29, 2025.

This paper was recommended for publication by Editor Lucia Pallottino upon evaluation of the Associate Editor and Reviewers' comments.

This work was in part supported by the National Science Foundation grants 1932187, 2026479, 2222815 and 2315396 and ANITI (ANR 19-P3IA-0004). This work was supported by a grant overseen by the French National Research Agency (ANR) and France 2030 as part of the PR[AI]RIE-PSAI AI cluster (ANR-23-IACL-0008), by the French government under the management of Agence Nationale de la Recherche through the project NIMBLE (ANR-22-CE33-0008), and by the European Union through the AGIMUS project (GA no.101070165).

<sup>1</sup> Machines in Motion Laboratory, New York University, USA

<sup>2</sup> Inria - Département d'Informatique de l'École normale supérieure, PSL Research University.

<sup>3</sup> LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

<sup>4</sup> Inria, AUCTUS team, Talence, France

<sup>5</sup> Artificial and Natural Intelligence Toulouse Institute (ANITI), Toulouse

\* Corresponding author: armand.jordana@nyu.edu

Digital Object Identifier 10.1109/LRA.2025.3577875

Reinforcement Learning (RL) [8] and Approximate Dynamic Programming (ADP) [7] appear as good candidates to move the compute time offline by approximating policies or value functions. Those techniques recently have shown impressive results in locomotion [9], [10] and manipulation [11], [12]. However, they are subject to the curse of dimensionality and can be unsafe outside of the training distribution. Also, incorporating safety by imposing hard constraints is challenging with current RL and ADP tools. In contrast, by re-planning online, constrained MPC can adapt to novel situations and ensure hard constraint satisfaction.

Combining the advantages of online and offline decision making is appealing, and it has proven to be highly effective in the context of games such as Go or Chess [13]. In the context of robotics, it could allow maintaining safety by ensuring hard constraints while leveraging the full potential of function approximation. In this work, we propose to combine MPC with function approximation in order to perform infinite horizon constrained MPC. The infinite horizon value function is approximated using neural networks and trained using value iteration and local gradient-based optimization. Then, during deployment, the approximated value is used as a terminal cost for the Optimal Control Problem (OCP). We show experimentally that the approximated value function helps the controller escape local minima while online optimization compensates for approximation errors. Furthermore, outside of the training distribution, the controller might not be optimal but, online optimization ensures hard constraint satisfaction given a control-invariant set.

The theoretical benefits of an infinite horizon formulation have been extensively studied [6], [14]–[16]. Unfortunately, the general case is intractable. Hence, efforts have concentrated on the constrained linear quadratic regulator [17]–[19]. In the general case, RL provides a way to approximate the solution [7], [8]. In the discrete action setting, Deep Q-learning is a popular tool [20]. In the continuous case, actor-critics such as DDPG [21] or SAC [22] allow to learn simultaneously a policy and a value function. However, despite recent progress [23], [24], incorporating hard constraints in those formulations remains challenging. Also, RL algorithms [8] typically use a discount factor. However, the global stability guarantees of infinite horizon MPC were established in the non-discounted setting [5], [6]. Consequently, we study the non-discounted setting with hard constraints which, to the best of our knowledge, remains understudied in the RL community.

The idea of combining MPC and function approximation is not novel and has fostered a lot of research in the robotics community. The seminal work of Atkeson [25] explored how

**IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.**

to use local trajectory optimization together with a global value function. Since then, an extensive amount of work has shown the benefits of using TO with learning to either speed up the training of value functions and policies or improve the controller's performance at test time [26]–[36]. A limitation of these works is their inability to consider hard nonlinear constraints. In practice, constraints have to be enforced softly using penalty terms in the cost function. However, this approach requires tedious weight tuning and can hardly provide guarantees. In this work, we show how the known advantages of combining MPC and RL can be obtained while enforcing hard constraints.

More recently, [37], [38] demonstrated on hardware the benefits of using online constrained optimization with an approximate value as a terminal model in the context of locomotion. In these works, the authors propose to learn the value with a local TO solver using a long horizon. Consequently, at test time, the controller remains local. In contrast, we use value iteration combined with local TO to approximate the infinite horizon value function and we demonstrate the ability of the method to avoid local minima.

In this work, we propose to approximate the infinite horizon value function of constrained OCP and use it as a terminal cost function of a Model Predictive Controller. The contributions of this paper are threefold:

- First, we demonstrate how a local gradient-based solver allows the use of value iteration to approximate the optimal value function of an infinite horizon constrained OCP.
- Second, we provide an experimental study showing how the use of trajectory optimization can compensate for the inaccuracies of the value function approximation.
- Third, we demonstrate the benefits of combining MPC with value function approximation on a reaching task with obstacle avoidance on an industrial manipulator. More precisely, we show experimentally how the use of the value function allows avoiding the local minima MPC is subject to. Furthermore, given a control-invariant set, we demonstrate how the method remains safe by ensuring hard constraints outside the training distribution of the value function. To the best of our knowledge, this is the first demonstration of MPC using a learned infinite horizon value function with hard constraints deployed on a robot at real-time rates.

## II. BACKGROUND

In this work, we are interested in the infinite-horizon constrained optimal control problem:

$$V(x) = \lim_{T \rightarrow \infty} \min_{u_0, u_1, \dots, u_{T-1}} \sum_{k=0}^{T-1} \ell(x_k, u_k) \quad (1a)$$

$$\text{s.t. } x_0 = x \quad (1b)$$

$$x_{k+1} = f(x_k, u_k) \quad (1c)$$

$$c(x_k, u_k) \geq 0 \quad (1d)$$

Here  $V$  is the infinite horizon value function. The state  $x$  belongs to  $\mathbb{R}^{n_x}$  and the control  $u$  to  $\mathbb{R}^{n_u}$ . The dynamics

function  $f$  maps  $\mathbb{R}^{n_x} \times \mathbb{R}^{n_u}$  to  $\mathbb{R}^{n_x}$ . The constraint function  $c$  maps  $\mathbb{R}^{n_x} \times \mathbb{R}^{n_u}$  to  $\mathbb{R}^{n_c}$ . The cost function  $\ell$  maps  $\mathbb{R}^{n_x} \times \mathbb{R}^{n_u}$  to  $\mathbb{R}^+$ . Similarly to [39], to ensure the existence of states yielding a finite value function, we consider that:

**Assumption.** *The set of stationary points yielding a zero cost*

$$\mathcal{G} = \{x \mid \exists u \text{ s.t. } \ell(x, u) = 0, x = f(x, u), c(x, u) \geq 0\} \quad (2)$$

*is not empty.*

Due to the infinite sum, computing the value function or its associated optimal policy is intractable in general. However, RL provides tools to find an approximation.

Let's denote  $\Omega$ , the space on which the value function is well-defined. That is to say the set of initial states such that there exists a control sequence that ensures constraint satisfaction for all future states. For any state  $x \in \Omega$ , the value function satisfies the Bellman equation:

$$\mathcal{B}(V) = V, \quad (3)$$

where  $\mathcal{B}$  is the Bellman operator, defined by:

$$\mathcal{B}(V)(x) = \min_u \ell(x, u) + V(f(x, u)) \quad (4a)$$

$$\text{s.t. } c(x, u) \geq 0 \quad (4b)$$

$$f(x, u) \in \Omega \quad (4c)$$

Note that by definition,  $\Omega$  is a control-invariant set. Hence, Eq (4c) ensures recursive feasibility [6] and safety [40]. In this work, we focus on problems where the set  $\Omega$  is known and can be expressed analytically, which encompasses many problems. For instance, if the constraint is of the form  $c(u)$ , then  $\Omega = \mathbb{R}^{n_x}$ . For a fully actuated system, if the constraint is of the form  $c(x)$  where  $x$  denotes position and velocity and  $u$  denotes torque control inputs, then  $\Omega = \{x \in \mathbb{R}^{n_x} \mid c(x) \geq 0\}$ . In the more general case where the dynamics are nonlinear and the constraint is a function of both the state and the control, the set  $\Omega$  is not tractable, and we would have to rely on approximation techniques [41], [42]. However, this is beyond the scope of the paper.

In the non-discounted setting, the Bellman equation generally has multiple solutions since adding a positive constant to any solution produces another solution [39]. However, the fact that points in  $\mathcal{G}$  yield a zero cost can allow us to recover the value function defined in Eq (1). In fact, a necessary and sufficient condition to recover the value function defined in Eq (1) is to find a function that satisfies the Bellman equation and that yields a zero value on stationary points [39]. In other words, we need to find a function  $V$  satisfying the Bellman equation (4) and such that,

$$\forall x \in \mathcal{G}, \quad V(x) = 0. \quad (5)$$

Value iteration provides a way to find a solution to the Bellman equation by iteratively applying the Bellman operator:

$$V_{k+1} = \mathcal{B}(V_k). \quad (6)$$

Under the assumption that the set  $\mathcal{G}$  is not empty, it can be shown that  $V_k$  converges to  $V$  pointwise [39], [43].

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

### III. METHOD

#### A. $T$ -step optimal lookahead problem

In this work, we propose to combine online and offline decision-making by using the  $T$ -step optimal lookahead problem [44] online. The idea is to perform MPC by solving an OCP at each time step, using an approximate value function as a terminal cost function. More precisely, given a state  $x$ , we aim to find the optimal action by solving:

$$\min_{u_0, u_1, \dots, u_{T-1}} \sum_{k=0}^{T-1} \ell(x_k, u_k) + V_\theta(x_T) \quad (7a)$$

$$\text{s.t. } x_0 = x \quad (7b)$$

$$x_{k+1} = f(x_k, u_k) \quad (7c)$$

$$c(x_k, u_k) \geq 0 \quad (7d)$$

$$x_T \in \Omega \quad (7e)$$

Here,  $V_\theta$  is the value function approximation. In this work, we consider  $V_\theta$  to be a neural network parameterized by weights  $\theta$ . At each time step, the first optimal control input,  $u_0^*$ , is applied to the system, and the other controls are disregarded. In the end, the policy,  $\pi^*(x) = u_0^*$ , depends on the horizon  $T$  and the approximated value function  $V_\theta$ . In continuous action space, RL algorithms rely on a function approximation of the policy [8]. In contrast, we solve Problem (7) online. The benefit of this approach is that the model used during the first  $T$  steps of the optimization can reduce the inaccuracies of the value function and guarantee stability [45]–[47]. Furthermore, given knowledge of  $\Omega$ , this ensures hard constraint satisfaction despite the use of an approximated value function.

#### B. Value function approximation

In this section, we show how to use value iteration defined in Equation (6) to approximate the value function of an infinite-horizon constrained OCP by neural networks. To minimize the Bellman Equation (4), we propose to use a local gradient-based solver. To reduce the required number of value iterations (i.e. iterations of the Bellman operator), we apply the minimization over an arbitrary horizon  $T$ . More precisely, we use trajectory optimization to directly solve:

$$V_{k+1} = \mathcal{B}^{[T]}(V_k) \quad (8)$$

where  $\mathcal{B}^{[T]}$  is the Bellman operator over a horizon  $T$ .

$$\mathcal{B}^{[T]}(V) = \underbrace{\mathcal{B} \circ \dots \circ \mathcal{B}}_{T \text{ times}}(V) \quad (9)$$

Indeed, it can be shown by induction that iterating  $T$  times the Bellman operator is equivalent to solving an OCP of horizon  $T$ , precisely as in Problem (7). Intuitively, this should allow us to perform  $T$  times fewer value iterations.

In this work, we propose to fit the approximated value function at each Bellman iteration in a supervised way. Note that this approach can be considered as an instance of Fitted Value Iteration (FVI) [7], [48] adapted to the constrained and deterministic setting. More precisely, at iteration  $k$ , given a value function  $V_k$ , we sample  $n$  states,  $\{x_j\}_{1 \leq j \leq n}$  and solve the  $n$  associated OCP with the corresponding initial condition

and  $V_k$  as a terminal cost. Then, we train in a supervised way such that  $V_\theta$  maps  $x_j$  to  $\mathcal{B}^{[T]}(V_k)(x_j)$ . Furthermore, to ensure that  $\forall x^s \in \mathcal{G}, V_\theta(x^s) = 0$ , we sample  $m$  stationary points,  $\{x_j^s\}_{1 \leq j \leq m}$ , and minimize the value function at those points with the Mean Squared Error (MSE). In the end, we minimize the following loss:

$$\sum_{j=1}^n \left( V_\theta(x_j) - \mathcal{B}^{[T]}(V_k)(x_j) \right)^2 + \alpha \sum_{j=1}^m V_\theta(x_j^s)^2, \quad (10)$$

where  $\alpha$  is a penalty parameter. To create the targets, we solve Problem (7) using the stagewise Sequential Quadratic Programming (SQP) implementation introduced in [49]. This solver can handle hard constraints and exploits the time sparsity of the problem by using Riccati recursions in order to guarantee a linear complexity with the time horizon and quadratic convergence. Algorithm 1 summarizes the method.

---

#### Algorithm 1: Value Iteration

---

**Input:** dynamics  $f$ , cost  $\ell$ , constraint  $c$ , horizon  $T$ , network parameters  $\theta$

```

1 Initialize  $V_1$ 
  /* Main value iteration loop */
2 for  $k \leftarrow 1$  to  $N$  do
  /* Generate data */
3   for  $j \leftarrow 1$  to  $M$  do
4     Sample  $x_j$ 
5     Compute  $\mathcal{B}^{[T]}(V_k)(x_j)$  by optimizing (7)
6   Create dataset:  $\mathcal{D} = \{(x_j, \mathcal{B}^{[T]}(V_k)(x_j))_j\}$ 
  /* Fit value function with SGD */
7   for  $j \leftarrow 1$  to  $P$  do
8     Sample batch from  $\mathcal{D}$ 
9     Sample  $m$  stationary points
10    Apply gradient descent with loss (10);
11   $V_{k+1} \leftarrow V_\theta$ 

```

**Output:**  $V_{N+1}$

---

Minimizing (7) with a gradient-based solver requires the derivatives of the neural network. More specifically, the SQP approach requires the gradient and Hessian of the terminal cost function. To circumvent deriving twice a neural network, we use the Gauss-Newton approximation and define the value function as the squared L2 norm of a residual, i.e:

$$V_\theta(x) = \frac{1}{2} \|r_\theta(x)\|_2^2 \quad (11)$$

where  $r_\theta$  is a neural network with outputs in  $\mathbb{R}^d$ , where  $d$  is a hyperparameter. Using the Gauss-Newton approximation, we have:

$$\partial_x V_\theta(x) = \partial_x r_\theta(x)^T r_\theta(x) \quad (12a)$$

$$\partial_{xx}^2 V_\theta(x) \approx \partial_{xx} r_\theta(x)^T \partial_x r_\theta(x) \quad (12b)$$

where  $\partial r_\theta(x) \in \mathbb{R}^{d \times n_x}$  is the Jacobian matrix of  $r_\theta$  evaluated in  $x$ . This formulation is computationally efficient and also has the advantage of encoding the positiveness of the value function. For numerical optimization to be performed efficiently, it is crucial to obtain an accurate Jacobian of the value function.

**IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.**

While [34] investigated the use of Sobolev learning [50], we found that using  $\tanh$  activation functions and an appropriate weight decay was sufficient to ensure convergence of the SQP in few iterations. Furthermore, we find that initializing the initial guess on the value function,  $V_1$ , to the zero function improves the training. To do so, at the first value iteration, we solve the OCP without a terminal cost function.

In order to generate trajectories that are similar to the one encountered at deployment, we rollout trajectories to generate more data. More specifically, after solving Problem (7), the first state of the trajectory,  $x_1$ , is added to the dataset by solving Problem (7) using  $x_1$  as an initial condition, and we iterate until either the goal or a maximum number of iterations is reached. In other words, Problem (7) is used as an MPC controller to collect additional data. We find this especially relevant in complex examples with high-dimensional state space.

#### IV. EXPERIMENTS

In this section, we present three problems of increasing complexity. The first two examples are used to illustrate the ability of the method to approximate the infinite horizon problem with a finite horizon and an approximate terminal value function. Lastly, we study a reaching task on an industrial manipulator with an obstacle to demonstrate the scalability of the method. In order to handle a moving scene, the value function is conditioned on the goal of the reaching task and the obstacle pose. First, we provide an analysis of the impact of the horizon both at train and test time. Then, we present results from real experiments. For all experiments, we use the SQP implementation introduced in [49]. During the training, we solve each OCP in parallel on the CPU. Note that this is crucial to obtain reasonable training times.

##### A. Toy Problem 1: Constrained Simple Pendulum

The first test problem we consider is the swing-up of a simple pendulum with torque limits. We illustrate how a finite horizon with an approximate value function as a terminal cost allows us to approximate the infinite horizon MPC. The state is  $x = [\theta, \dot{\theta}]^T$  where  $\theta$  denotes the orientation of the pendulum. The dynamics are defined by applying Euler integration to the following law of motion:

$$\ddot{\theta} = -\frac{g}{L} \sin(\theta) \quad (13)$$

The goal is to bring the pendulum to the upward position, which is incentivized with the following cost:

$$\ell(x, u) = \cos(\theta) + 1 + 0.01\dot{\theta}^2 + 0.001u^2 \quad (14)$$

Lastly, the control input,  $u$ , is constrained to be within  $[-2, 2]$  which makes it impossible to swing up the pendulum without several back and forth. We sample  $\theta$  uniformly in  $[-\pi, \pi]$  and  $\dot{\theta}$  uniformly in  $[-6, 6]$ . The number of sample points at each value iteration,  $n$ , is set to 500, and the number of goals sampled  $m$  is set to 1 as  $\mathcal{G}$  is a singleton. We consider a horizon of length  $T = 10$  and perform 1000 value iterations. At each iteration, we perform 80 SGD steps with  $\alpha = 1$  using

Adam with default parameters and a weight decay of  $10^{-4}$ . The network is a three-layer MLP with 64 neurons and output size  $d = 64$ . In the end, the training lasts 2 minutes and 40 seconds.

Figure 1 shows the behavior of the MPC controller for different horizon lengths using the approximated value function as a terminal cost. The quality of the control increases as the horizon length increases. This can also be seen by looking at the final cumulative cost. The longer the horizon, the more optimal the controller is. For  $T = 1$ , the cost is 4.57, for  $T = 10$ , the cost is 4.50 and for  $T = 20$ , the cost is 4.41. This illustrates how solving online Problem (7) compensates for the approximation error of the value function.

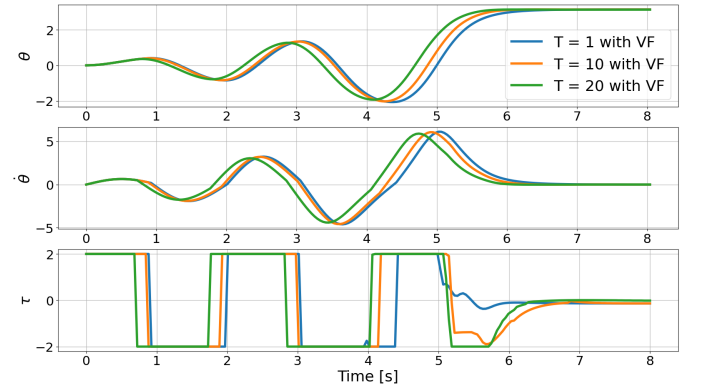


Fig. 1: Rollout of MPC controllers with different horizon lengths using the learned value function as a terminal cost.

##### B. Toy Problem 2: Constrained point

In this second toy example, we illustrate the ability of the method to avoid local minima to which MPC is prone. We consider a 2-dimensional point that has to move around an obstacle to reach a target. The state,  $x$ , denotes the 2D position, and the control  $u$  denotes the velocity of  $x$ . The dynamics and cost are defined in the following way:

$$x_{t+1} = x_t + \Delta t u_t \quad (15)$$

$$\ell(x, u) = \|x - x^*\|_2^2 + 0.1\|u\|_2^2 \quad (16)$$

where  $\Delta t$  is set to 0.02. The constraints are defined by the distance between the point and the obstacle, as illustrated in Figure 2. We sample  $x$  uniformly and reject states inside the obstacle. The number of samples,  $n$ , is set to 2500, and we augment the dataset with the last state of each trajectory. We chose  $m = 1$  as  $\mathcal{G}$  is a singleton. We consider a horizon of 10 and perform 100 value iterations using  $\alpha = 1$ . At each iteration, we perform 2000 SGD steps using Adam with default parameters and a weight decay of  $10^{-4}$ . The network is a three-layer MLP with 32 neurons and output size  $d = 32$ . The training lasts 12 minutes. The time increase compared to the previous experiment is due to the implementation of the model in Python. Figure 2 shows how using the learned value function as a terminal cost allows to bypass the obstacle to reach the goal. Without the value function, the controller gets stuck in the corner of the obstacle.

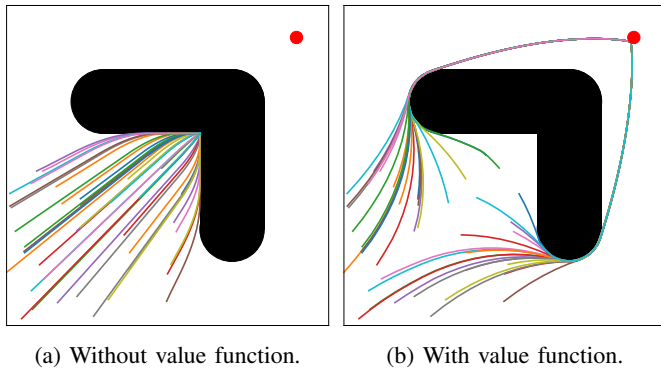


Fig. 2: MPC trajectories for different initial conditions. The red dot represents the target  $x^*$ .

### C. Manipulator experiment: influence of the horizon

*a) At train time:* In this section, we study the influence of the horizon on the training of the value function. We consider a reaching task with the 7-DoF Kuka iiwa robot. We consider an unconstrained OCP as in that setting, we can approximate the infinite sum in Eq. (1) with a large horizon  $T$ . Consequently, we use our solver with  $T = 200$  (and no terminal cost) to approximate the ground truth infinite horizon value function. We fix the last joint and consider a 12-dimensional state containing joint positions and velocities. The OCP includes an end-effector target reaching cost, joint velocity regularization, and joint torque regularization costs. At each value iteration, we collect 10000 trajectories of length 10 by sampling the initial configuration uniformly within the joint and velocity bounds of the robot. Then, we perform 16 epochs. Figure 3 shows the MSE between the learned value and the ground truth during training. The larger the horizon in Eq (7) is, the faster the algorithm converges to the ground truth. Note that the data generation time is negligible compared to the time required by SGD to fit the network. Therefore, the total training time is similar for all horizon lengths.

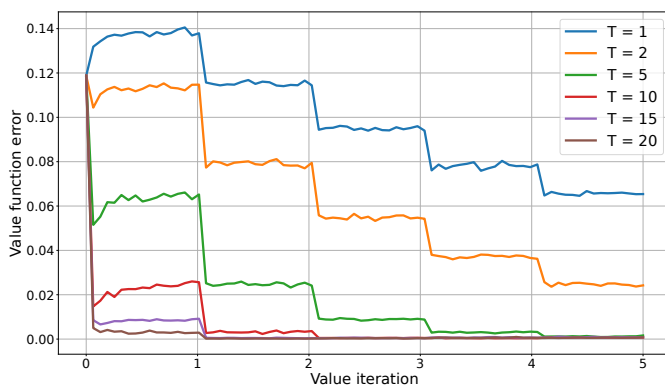


Fig. 3: Error between the ground truth and the learned value function during training for various horizon length. The larger the horizon is, the faster the algorithm converges.

*b) At test time:* In this study, we show that online optimization can compensate for the value function approximation error due to learning. We use the same setup as in the previous section and show that at test time, a longer horizon helps to reduce the running cost. We perform value iteration with a

horizon of 10 and use the same parameters as in the previous section. Furthermore, we illustrate that the improvement due to the horizon is not specific to our training procedure but due to the limitation of the neural network’s expressivity. To do so, we train in a supervised way the value function with 100000 ground truth MPC trajectories of length 10. We use the same number of SGD steps as in the overall value iteration learning procedure; therefore, the test time performance of this network can be considered as an upper bound on the one of the value iteration network. Lastly, using the ground truth data, we train in a supervised way a policy mapping states to torques (while removing the gravity compensation). Figure 4 shows the cost error between various controllers and the ground truth infinite horizon controller. It can be seen that for both value functions, increasing the horizon improves performance. Also, value iteration training achieves a performance close to supervised training which is provided with the ground truth values.

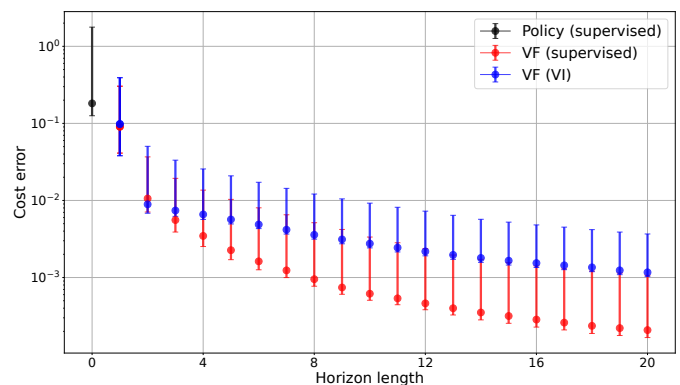


Fig. 4: We run 1000 MPC simulations starting from random initial states with increasing horizon for each controller. Horizon 0 corresponds to the policy.

### D. Real manipulator experiments

*1) Setup:* We validate the proposed approach on the KUKA iiwa LBR 1480 in target reaching/tracking and obstacle avoidance tasks. We use a motion capture system (VICON) to track the targets and obstacles. The robot receives joint torque commands and a PD joint state reference at 1 kHz through the FRI. The overall control law reads

$$\tau = u_0^* - K_P(\hat{q} - q_1^*) - K_D(\hat{q} - \dot{q}_1^*) \quad (17)$$

where  $u_0^*$  and  $x_1^*$  are the optimal control input and the predicted state respectively (computed by the MPC at 100 Hz) and  $\hat{x}$  is the measured state of joint positions and velocities, controlled at 1 kHz by the joint PD. We use  $K_P = [150, 150, 100, 100, 50, 10, 10]$  and  $K_D = [25, 25, 20, 20, 14, 6, 6]$ . The measured position is directly read from the robot’s encoders, while the velocity is estimated by finite differences. Note that the last joint is blocked and not part of the model, as it speeds up learning and is not necessary for the tasks under study.

The obstacle we consider is a thin rod of length 76 cm. To enforce collision avoidance, we cover the robot with

**IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.**

capsules and define the signed distance between the rod and the capsules, as well as between the table and the capsules, as hard constraints [51]. In total, this represents 14 constraints.

To deploy the method on the real system, we condition the network on both the target position and obstacle pose. States are sampled within 70% of the robot’s joint and velocity range. The target’s Cartesian position is sampled within  $[0.45, 0.75] \times [-0.2, 0.2] \times [0.15, 0.5]$  and states with end-effector positions outside this region are discarded, as these are outside the robot’s workspace. Obstacle poses are sampled within a 10 cm cube in front of the robot. We also randomize slightly the orientation through a uniform sampling of the Euler angles within  $[-0.1, 0.1]$ . Lastly, we reject the triplets (state, target, obstacle) with collisions between the robot and the obstacle or for which inverse kinematics has no solution for the given target. Additionally, to better reflect scenarios encountered on the real system, we reject triplets where both the target and end-effector are above the rod. We consider  $T = 5$  and perform 500 value iterations with  $\alpha = 0.01$ . At each iteration, we collect a dataset by sampling 1000 triplets and rolling out trajectories up to 60 time steps or until the robot reaches the target. The target is considered to be reached whenever the running cost is below 0.1. Then, we perform 16 epochs using Adam with a learning rate of 0.0004 and a weight decay of  $10^{-5}$ . The training lasts 3 h 20 min on CPU only.

During deployment, the learned value function is used as a terminal cost in the MPC, while the baseline MPC has no terminal cost. The horizon used is  $T = 10$ , with an OCP discretization of  $\Delta T = 50$  ms. The maximum number of SQP iterations is set to 6, the termination tolerance to  $10^{-4}$ , and the maximum number of QP iterations to 200. The OCP includes the non-collision constraints and state limits.

We use a 4-layer MLP with 64 neurons per layer and an output size  $d = 64$ . To minimize compute time, the network and its derivative are implemented in C++. For our architecture, the inference time is similar to the one of the model (19  $\mu$ s for the network and 16  $\mu$ s for the model). In other words, the neural network does not limit the horizon length we can consider during real-time experiments.

2) *Pick-and-place with static obstacle*: We compared the performance of the proposed approach against the default MPC on a pick-and-place task with a static obstacle. The robot must alternatively reach two end-effector positions while avoiding collision with a fixed rod laying in between the targets. Figure 5 shows that the MPC with value function reaches the target (by finding a path moving the end-effector *above* the rod) while the default MPC remains stuck in a local minimum (trying to go *underneath* the rod). This behavior can be further understood by looking at the total cost after 10 seconds. With the value function, the final cost is 213 while it reaches 1311 without it. The MPC with value function achieves a lower cost since it eventually reaches the target. Interestingly, this controller initially increases its cost faster than the default MPC. This is due to higher velocity and torque regularization cost residuals during the obstacle avoidance motion. Hence, this experiment shows the ability of the proposed controller to avoid local minima thanks to the value function. Indeed, the default MPC remains stuck in a

local minimum and must trade off the task completion against constraint satisfaction. In contrast, our approach is able to both achieve the task *and* avoid collision with the rod by choosing a different path that initially increases the cost. It is important to remind that both controllers use the same warm-start, the only difference is the terminal cost used.

3) *Target tracking with static obstacle*: In this experiment, we show the ability of our approach to track a moving target while satisfying obstacle avoidance constraints. We use a small cube tracked by the motion capture system to define a moving target. The accompanying video <sup>1</sup> shows how the controller can go around the obstacles when the cube is moved from one side to the other of the obstacle. Figure 6 depicts the constraint satisfaction between the rod and the capsules. The negative values indicate small constraint violations which can be explained by the fact that the solver does not always reach full convergence. In practice, the capsules are slightly larger than the robot, and the constraint violations are small enough for the robot to avoid collisions on the hardware.

4) *Target tracking with dynamic obstacle*: Lastly, we illustrate in the video the ability of the method to deal with out-of-distribution orientation of the rod as well as unexpected disturbances. Intuitively, far from the training distribution, the approximated value function is not very meaningful and cannot provide a way to avoid the obstacle. In fact, it can be seen that the controller is no longer able to systematically avoid local minima due to the obstacle. However, the hard constraints are satisfied.

## V. DISCUSSION

While value iteration is guaranteed to converge [39], this relies on the assumption that the minimization in Problem (7) is global. However, we use a gradient-based solver which can be subject to local minima. Intuitively, the larger  $T$  is, the more the solver is prone to local minima. In practice, we found that tuning appropriately  $T$  was enough to get VI to converge. In the end, the choice of  $T$  regulates the trade-off between the efficiency of the local solver and the global property of value iteration. Lastly, we have investigated the use of various random warm-starts to search for the global solution; however, the convergence speed-up did not compensate for the additional computational time. Nevertheless, this remains an interesting direction to explore further.

Another limitation of our method is that gradient-based solvers require a smooth neural network. This constrains the network architecture and training parameters. More specifically, the SQP solver could not handle a network with ReLU activation and required an appropriate tuning of the weight decay. Without weight decay, the network would overfit and the solver’s number of iterations would diverge. To circumvent the issue, it would be interesting to investigate the use of zero-order methods, which have recently shown promising results [52], [53].

One of the key assumptions of the work is the tractability of the feasible set  $\Omega$ . Although this assumption encompasses a wide set of problems, it would be interesting to study how to

<sup>1</sup><https://youtu.be/CruTx2CvcFQ>

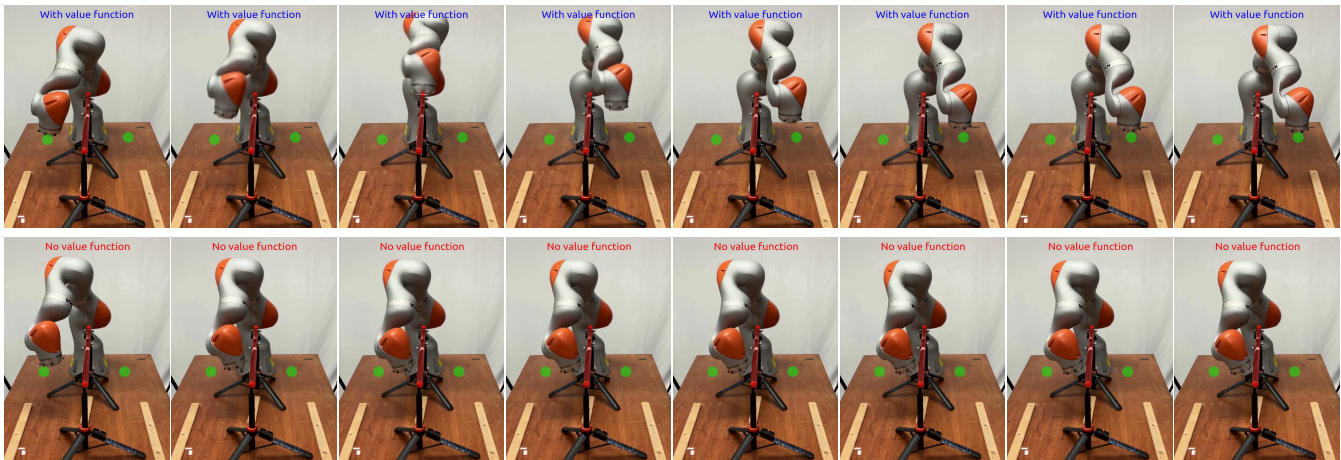


Fig. 5: Snapshots of pick-and-place task with static obstacle avoidance for the default MPC without value function (bottom) and the proposed MPC with value function (top). The green dots represent the end-effector targets that must be reached alternatively while avoiding collision with the black rod placed in the center (highlighted in red).

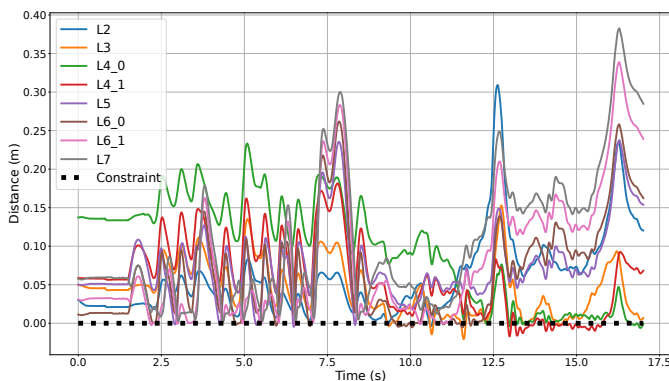


Fig. 6: Target tracking with static obstacle. Collision signed distance between the robot’s capsules and the obstacle.

generalize to any type of constraint. A naive approach could be to first approximate  $\Omega$  with other methods such as [41], [42] and then apply value iteration. However, it would be interesting to combine those two steps.

This work focuses on the non-discounted setting because this is the original MPC formulation that can guarantee stability [5], [6]. Arguably, using a non-discounted setting requires a cautious design of the problem as it is crucial to ensure that the goal states achieve zero cost. Furthermore, while [39], [43] proved the convergence of value iteration in the non-discounted setting, it is still not clear how to guarantee convergence while approximating the value with neural networks. In contrast, popular RL algorithms usually use a discount factor [8]. It would be interesting to study the impact of the discount parameter during training. However, it is not clear if the stability guarantees of the infinite horizon [6] will be preserved in that setting.

In the manipulator experiment, one limitation is that our training assumes that the goal and the obstacle do not move. If we knew their dynamics, we could add them to the state space and apply our method. However, this could make the set  $\Omega$  intractable. Lastly, for more complex problems, there might

be a trade-off between the neural network expressivity and inference time: a large network might increase the accuracy but also slow down the evaluation time, which may in turn limit the horizon length that can be used during deployment. However, in our experiments, the network architecture yielding the best approximation was not very large and its inference time was comparable to that of the model. As a result, we did not face this trade-off in practice.

## VI. CONCLUSIONS

We have introduced a way to combine constrained TO with RL by using a learned value function as a terminal cost of the MPC. We have demonstrated the benefits of the approach on a reaching task with obstacles on an industrial manipulator. In contrast to traditional MPC, by approximating an infinite horizon OCP, the method can avoid complex local minima. Furthermore, in contrast to RL, the online use of TO allows us to gain accuracy as it can leverage online the model of the robot. Furthermore, the proposed method allows us to handle hard constraints and out-of-distribution states. Future work will investigate how to extend this work to problems involving contacts such as locomotion and manipulation.

## REFERENCES

- [1] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, “Whole-body model-predictive control applied to the hrp-2 humanoid,” in *RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 3346–3351.
- [2] M. Neunert, M. Stäuble, M. Gifftthaler, 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.
- [3] J.-P. Sleiman, F. Farshidian, M. V. Minniti, and M. Hutter, “A unified mpc framework for whole-body dynamic locomotion and manipulation,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4688–4695, 2021.
- [4] E. Dantec *et al.*, “Whole-body model predictive control for biped locomotion on a torque-controlled humanoid robot,” in *International Conference on Humanoid Robots*. IEEE, 2022, pp. 638–644.
- [5] D. Q. Mayne, “Model predictive control: Recent developments and future promise,” *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.

## IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

- [6] L. Grüne and J. Pannek, *Nonlinear model predictive control*. Springer, 2017.
- [7] D. Bertsekas, *Reinforcement learning and optimal control*. Athena Scientific, 2019, vol. 1.
- [8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [9] A. Agarwal, A. Kumar, J. Malik, and D. Pathak, “Legged locomotion in challenging terrains using egocentric vision,” in *Conference on robot learning*. PMLR, 2023, pp. 403–415.
- [10] D. Hoeller, N. Rudin, D. Sako, and M. Hutter, “Anymal parkour: Learning agile navigation for quadrupedal robots,” *Science Robotics*, vol. 9, no. 88, p. eadi7566, 2024.
- [11] T. Chen, M. Tippur, S. Wu, V. Kumar, E. Adelson, and P. Agrawal, “Visual dexterity: In-hand reorientation of novel and complex object shapes,” *Science Robotics*, vol. 8, no. 84, p. eadc9244, 2023.
- [12] A. Handa *et al.*, “Dextreme: Transfer of agile in-hand manipulation from simulation to reality,” in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 5977–5984.
- [13] D. Silver *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [14] H. Chen and F. Allgöwer, “A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability,” *Automatica*, vol. 34, no. 10, pp. 1205–1217, 1998.
- [15] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, “Constrained model predictive control: Stability and optimality,” *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.
- [16] B. Hu and A. Linnemann, “Toward infinite-horizon optimality in nonlinear model predictive control,” *IEEE Transactions on Automatic Control*, vol. 47, no. 4, pp. 679–682, 2002.
- [17] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, “The explicit linear quadratic regulator for constrained systems,” *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.
- [18] P. Grieder, F. Borrelli, F. Torrisi, and M. Morari, “Computation of the constrained infinite time linear quadratic regulator,” *Automatica*, vol. 40, no. 4, pp. 701–708, 2004.
- [19] G. Stathopoulos, M. Korda, and C. N. Jones, “Solving the infinite-horizon constrained lqr problem using accelerated dual proximal methods,” *IEEE Transactions on Automatic Control*, vol. 62, no. 4, pp. 1752–1767, 2016.
- [20] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [21] T. Lillicrap, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.
- [22] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.
- [23] L. Zhang, L. Shen, L. Yang, S. Chen, B. Yuan, X. Wang, and D. Tao, “Penalized proximal policy optimization for safe reinforcement learning,” *arXiv preprint arXiv:2205.11814*, 2022.
- [24] E. Chane-Sane, P.-A. Leziart, T. Flayols, O. Stasse, P. Souères, and N. Mansard, “Cat: Constraints as terminations for legged locomotion reinforcement learning,” *arXiv preprint arXiv:2403.18765*, 2024.
- [25] C. Atkeson, “Using local trajectory optimizers to speed up global optimization in dynamic programming,” *Advances in neural information processing systems*, vol. 6, 1993.
- [26] M. Zhong, M. Johnson, Y. Tassa, T. Erez, and E. Todorov, “Value function approximation and model predictive control,” in *2013 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*. IEEE, 2013, pp. 100–107.
- [27] S. Levine and V. Koltun, “Guided policy search,” in *International conference on machine learning*. PMLR, 2013, pp. 1–9.
- [28] I. Mordatch and E. Todorov, “Combining the benefits of function approximation and trajectory optimization,” in *Robotics: Science and Systems*, vol. 4, 2014, p. 23.
- [29] M. Korda, D. Henrion, and C. N. Jones, “Controller design and value function approximation for nonlinear dynamical systems,” *Automatica*, vol. 67, pp. 54–66, 2016.
- [30] K. Lowrey, A. Rajeswaran, S. Kakade, E. Todorov, and I. Mordatch, “Plan online, learn offline: Efficient learning and exploration via model-based control,” *arXiv preprint arXiv:1811.01848*, 2018.
- [31] D. Hoeller, F. Farshidian, and M. Hutter, “Deep value model predictive control,” in *Conference on Robot Learning*. PMLR, 2020, pp. 990–1004.
- [32] B. Landry, H. Dai, and M. Pavone, “Seagull: Sample efficient adversarially guided learning of value functions,” in *Learning for Dynamics and Control*. PMLR, 2021, pp. 1105–1117.
- [33] N. Hatch and B. Boots, “The value of planning for infinite-horizon model predictive control,” in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 7372–7378.
- [34] A. Parag, S. Kleff, L. Saci, N. Mansard, and O. Stasse, “Value learning from trajectory optimization and sobolev descent: A step toward reinforcement learning with superlinear convergence properties,” *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1410–1416, 2022.
- [35] G. Grandesso, E. Alboni, G. P. R. Papini, P. M. Wensing, and A. Del Prete, “Cacto: Continuous actor-critic with trajectory optimization—towards global optimality,” *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3318–3325, 2023.
- [36] E. Alboni, G. Grandesso, G. P. R. Papini, J. Carpentier, and A. Del Prete, “Cacto-sl: Using sobolev learning to improve continuous actor-critic with trajectory optimization,” in *6th Annual Learning for Dynamics & Control Conference*. PMLR, 2024, pp. 1452–1463.
- [37] J. Viereck, A. Meduri, and L. Righetti, “Valuenetqp: Learned one-step optimal control for legged locomotion,” in *Learning for Dynamics and Control Conference*. PMLR, 2022, pp. 931–942.
- [38] J. Wang *et al.*, “Online multi-contact receding horizon planning via value function approximation,” *IEEE Transactions on Robotics*, 2024.
- [39] D. P. Bertsekas, “Value and policy iterations in optimal control and adaptive dynamic programming,” *IEEE transactions on neural networks and learning systems*, vol. 28, no. 3, pp. 500–509, 2015.
- [40] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, “Control barrier functions: Theory and applications,” in *18th European control conference (ECC)*. Ieee, 2019.
- [41] B. Djeridane and J. Lygeros, “Neural approximation of pde solutions: An application to reachability computations,” in *Proceedings of the 45th IEEE Conference on Decision and Control*, 2006, pp. 3034–3039.
- [42] S. Bansal and C. J. Tomlin, “Deepreach: A deep learning approach to high-dimensional reachability,” in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 1817–1824.
- [43] A. Heydari, “Revisiting approximate dynamic programming and its convergence,” *IEEE transactions on cybernetics*, vol. 44, no. 12, pp. 2733–2743, 2014.
- [44] D. Bertsekas, *Dynamic programming and optimal control: Volume I*. Athena scientific, 2012, vol. 4.
- [45] D. P. Bertsekas, “Dynamic programming and suboptimal control: A survey from adp to mpc,” *European journal of control*, vol. 11, no. 4-5, pp. 310–334, 2005.
- [46] A. J. Krener, “Adaptive horizon model predictive control and al’brekht’s method,” in *Encyclopedia of Systems and Control*. Springer, 2021, pp. 27–40.
- [47] D. P. Bertsekas, “Model predictive control and reinforcement learning: A unified framework based on dynamic programming,” *IFAC-PapersOnLine*, vol. 58, no. 18, pp. 363–383, 2024.
- [48] R. Munos and C. Szepesvári, “Finite-time bounds for fitted value iteration,” *Journal of Machine Learning Research*, vol. 9, no. 5, 2008.
- [49] A. Jordana, S. Kleff, A. Meduri, J. Carpentier, N. Mansard, and L. Righetti, “Stagewise implementations of sequential quadratic programming for model-predictive control,” *Subm. IEEE TRO*, 2023.
- [50] W. M. Czarnecki, S. Osindero, M. Jaderberg, G. Swirszcz, and R. Pascanu, “Sobolev training for neural networks,” *Advances in neural information processing systems*, vol. 30, 2017.
- [51] A. Haffemayer, A. Jordana, M. Fourny, K. Wojciechowski, G. Saurel, V. Petrik, F. Lamiroux, and N. Mansard, “Model predictive control under hard collision avoidance constraints for a robotic arm,” in *21st International Conference on Ubiquitous Robots (UR)*. IEEE, 2024.
- [52] A. H. Li, P. Culbertson, V. Kurtz, and A. D. Ames, “Drop: Dexterous reorientation via online planning,” *preprint arXiv:2409.14562*, 2024.
- [53] H. Xue, C. Pan, Z. Yi, G. Qu, and G. Shi, “Full-order sampling-based mpc for torque-level locomotion control via diffusion-style annealing,” *arXiv preprint arXiv:2409.15610*, 2024.