VBOC: Learning the Viability Boundary of a Robot Manipulator Using Optimal Control

Asia La Rocca, Matteo Saveriano, *Senior Member, IEEE*, and Andrea Del Prete, *Member, IEEE*

***Abstract*—Safety is often the most important requirement in robotics applications. Nonetheless, control techniques that can provide safety guarantees are still extremely rare for nonlinear systems, such as robot manipulators. A well-known tool to ensure safety is the viability kernel, which is the largest set of states from which safety can be ensured. Unfortunately, computing such a set for a nonlinear system is extremely challenging in general. Several numerical algorithms for approximating it have been proposed in the literature, but they suffer from the curse of dimensionality. This letter presents a new approach for numerically approximating the viability kernel of robot manipulators. Our approach solves optimal control problems to compute states that are guaranteed to be on the boundary of the set. This allows us to learn directly the set boundary, therefore learning in a smaller dimensional space. Compared to the state of the art on systems up to dimension 6, our algorithm resulted to be more than 2 times as accurate for the same computation time, or 6 times as fast to reach the same accuracy.**

***Index Terms*—Control system security, optimal control, robot control, robot learning.**

I. Introduction

THE computation of viability kernels is a topic of great importance in the field of safe control of constrained dynamical systems. The viability kernel is the set of states from which a dynamical system can remain within a predefined set of safe states. Knowing the viability kernel, it is straightforward to design safe controllers; it is therefore a very powerful tool for safety-critical applications. Unfortunately, except for the case of linear dynamics and linear constraints [1], [2], computing these sets is extremely challenging.

The classical method for the computation of these sets, known as the Viability Kernel Algorithm [3], consists in gridding the state space and approximating the viability kernel using recursive inclusions. Due to the grid-based discretization of the state space that is performed, the complexity and memory requirements of this algorithm scale exponentially with respect to the state dimension (problem known as the curse of dimensionality). The problem has then been approached with many different tools such as viability theory [4], theory of barriers [5], approximate dynamic programming [6], or simulated annealing [7].

Manuscript received 28 April 2023; accepted 23 August 2023. Date of publication 11 September 2023; date of current version 19 September 2023. This letter was recommended for publication by Associate Editor V. Modugno and Editor J. Kober upon evaluation of the reviewers' comments. This work was supported in part by the PRIN Project DOCEAT under CUP Grant E63C22000410001 and in part by European Union through NextGenerationEU project iNest under ECS Grant 00000043. *(Corresponding author: Andrea Del Prete.)*

The authors are with the Industrial Engineering Department, University of Trento, 38123 Trento, Italy (e-mail: asia.larocca@unitn.it; matteo.saveriano@unitn.it; andrea.delprete@unitn.it).

Digital Object Identifier 10.1109/LRA.2023.3313921

One of the most relevant directions is given by the approaches based on reachability analysis, exploiting the connection between viability kernels and reachable sets [8]. Reachability analysis consists in inferring the set of all states that are backward/forward reachable by a constrained dynamical system from a given target/initial set of states. For nonlinear systems, examples are given in [9], [10], [11], using interval arithmetic, or in [6], [12], [13], using dynamic programming (level set approaches based on the solution of the Hamilton-Jacobi PDE). These algorithms try to accurately define the set boundary, but they suffer from the curse of dimensionality, restricting their application to small systems.

In recent years, an improvement in the field has been brought by function approximators, such as Neural Networks (NNs), which allow to represent complex sets in a more memory-efficient way with respect to gridding [12], [13], [14]. These algorithms require less memory to run and store the resulting approximation, which represented one of the main bottlenecks of previous approaches.

Another recent trend consists in the use of Reinforcement Learning, as in [15], where the authors exploit Q-learning to compute reach-avoid sets. This method allows to compute a safe under-approximation of these sets, but its applicability is still limited due to the need to discretize the action space.

A promising approach consists then in data-driven methods that rely on the feasibility of Optimal Control Problems (OCPs). In [16] the authors approximated the forward invariant region of nonlinear systems using Support Vector Machines (SVMs). Their algorithm solves OCPs for initial conditions well distributed over the state space, and uses the feasibility results to train a classifier. This approach is applicable to a wide range of nonlinear systems, but it becomes intractable for large state spaces, since it requires too many samples to get good approximations.

A promising way to reduce the computational burden of data-driven methods is Active Learning (AL), a machine learning framework to iteratively select new data samples that are the most informative or representative. In a first study [17], the authors proposed an iterative algorithm to select the points nearest to the frontier of the learned SVM classifier to improve its accuracy. However, the approach is still hardly scalable because the number of samples in large dimensions can still grow exponentially; moreover, nonlinear SVMs training complexity scales more than quadratically with the number of samples.

Overall, the computation of viability kernels remains an active area of research, with ongoing efforts to develop more efficient and scalable algorithms. In this letter we propose a new approach for the approximation of viability kernels of robot manipulators. Instead of computing the set boundary by iteratively approaching it (as in Active Learning or Hamilton-Jacobi methods), we

directly compute states that are *exactly* on the boundary. We then use these states to train an NN that approximates the set. The main advantage of this approach is that it requires significantly less samples than other data-driven methods. While our approach is tailored to robot manipulators (or any fully-actuated multi-body system), many of our theoretical results hold for any smooth dynamical system. Our tests on systems with 2, 4 and 6 dimensional states show that it leads to faster and more accurate approximations than state-of-the-art approaches.

II. Preliminaries

A. Notation

- $\partial\mathcal{V}$ denotes the boundary of the set $\mathcal{V}$;
- $\text{int}(\mathcal{V})$ denotes the interior of the set $\mathcal{V}$;
- $\mathcal{X} \setminus \mathcal{V}$ denotes the set difference between $\mathcal{X}$ and $\mathcal{V}$;
- $\{x_i\}_0^N$ denotes a discrete-time trajectory given by the sequence $(x_0, \dots, x_N)$;
- x^+ denotes the next state, whenever x is used to denote the current state.

B. Problem Statement

Let us consider a discrete-time dynamical system with state and control constraints:

$$x_{i+1} = f(x_i, u_i), \qquad x \in \mathcal{X} \subset \mathbb{R}^n, \qquad u \in \mathcal{U} \subset \mathbb{R}^m, \quad (1)$$

where $\mathcal{X}$ and $\mathcal{U}$ are the closed and bounded sets of feasible states and control inputs. Our goal is to compute a numerical approximation of the *viability kernel* $\mathcal{V}$, which is the subset of $\mathcal{X}$ starting from which it is possible to keep the state in $\mathcal{X}$ indefinitely. Mathematically, we can define $\mathcal{V}$ as:

$$\mathcal{V} \triangleq \{x_0 \in \mathcal{X} \,|\, \exists \{u_i\}_0^\infty : x_i \in \mathcal{X}, u_i \in \mathcal{U}, \forall\, i = 0, \dots, \infty\}. \quad (2)$$

In the following, we assume that $\mathcal{V}$ is closed.

Assumption 1: We assume that $f(\cdot)$ be differentiable with respect to x, which implies:

$$||\text{eig}(\partial_x f(x, u))|| < \infty \qquad \forall\, x \in \mathcal{X}, u \in \mathcal{U}, \quad (3)$$

where the function $\text{eig}(\cdot)$ returns the eigenvalues of the given matrix.

Assumption 1 implies that our method cannot handle non-smooth systems, such as a robot that makes contact with a perfectly rigid environment. However, contacts can sometimes be modeled as visco-elastic, recovering then differentiability.

C. Backward Reachability vs Viability

Our goal is to approximate $\mathcal{V}$, which is the largest control-invariant set [1] (i.e., a set inside which you can remain indefinitely). However, for many applications (e.g., ensuring recursive feasibility of MPC [18], or designing a safety filter for a Reinforcement Learning algorithm [19]) we could settle for just any sufficiently large control-invariant set. Control-invariant sets can be computed using N-step backward reachability, i.e., computing the set of points from which a given set can be reached in N steps. In certain cases, backward reachability can also be used to compute $\mathcal{V}$, and in the following we assume that this is the case.

Assumption 2: Let us define $\mathcal{S}$ as the set containing all the equilibrium states of our system:

$$\mathcal{S} = \{x \in \mathcal{X} \,|\, \exists\, u \in \mathcal{U} : x = f(x, u)\}. \quad (4)$$

We assume that the ∞-step backward reachable set of $\mathcal{S}$ is equivalent to $\mathcal{V}$. In other words, that a state is viable if and only if from that state you can reach an equilibrium state.

We argue that this assumption is satisfied for most robot manipulators, which are our main focus. However, even if this assumption were not satisfied, our approach could still be used to compute control-invariant sets.

D. Data-Driven Learning

A state-of-the-art approach to numerically approximate backward reachable sets is to sample states $x^{sample} \in \mathcal{X}$ and verify whether, from there, it is possible to reach the target set [20], $\mathcal{S}$ for our application. This can be done by solving an OCP like the following one:

$$\begin{aligned} \underset{\{x_i\}_0^N, \{u_i\}_0^{N-1}}{\text{maximize}} \quad & 1 \\ \text{subject to} \quad & x_{i+1} = f(x_i, u_i) \quad \forall\, i = 0, \dots, N-1 \\ & x_i \in \mathcal{X},\, u_i \in \mathcal{U} \quad \forall\, i = 0, \dots, N-1 \\ & x_0 = x^{sample} \\ & x_N = x_{N-1} \end{aligned} \quad (5)$$

where $N \in \mathbb{N}$ is the time horizon, which must be sufficiently large to allow the system to reach, if possible, an equilibrium state from x^{sample}. If a solution of this problem is found, then we know that $x^{sample} \in \mathcal{V}$, and the whole state trajectory is in $\mathcal{V}$. Otherwise, we can assume that $x^{sample} \notin \mathcal{V}$, even though this is not necessarily the case because a solution may exist even if the solver was unable to find one. While potentially impactful, this issue is typically neglected by assuming that the solver can find a solution if one exists. This information is then used to train a classifier (e.g., SVM or NN) to distinguish viable and non-viable states.

This approach scales badly because it requires a dense sampling of $\mathcal{X}$ to accurately approximate $\mathcal{V}$. To reduce complexity, it has been coupled with Active Learning (AL), a technique to choose the most informative or representative values of x^{sample}. A first study on the application of AL for the computation of viability kernels was done in [17], where the authors proposed an algorithm based on iteratively testing the nearest points to the frontier of the currently learned SVM classifier. More advanced AL algorithms for reachable sets approximations can be found in [21], [22].

Instead of iteratively approaching $\partial\mathcal{V}$, the next section presents an approach to directly compute states that are *exactly* on $\partial\mathcal{V}$.

III. Viability-Boundary Optimal Control (VBOC)

To find states that are *exactly* on $\partial\mathcal{V}$, we solve a modified version of OCP (5), where the initial state x_0 is not completely

fixed, but it is optimized through a cost function:

$$\begin{aligned} \underset{\{x_i\}_0^N,\{u_i\}_0^{N-1}}{\text{maximize}} \quad & a^\top x_0 \\ \text{subject to} \quad & x_{i+1} = f(x_i, u_i) \quad \forall i = 0, \dots, N-1 \\ & x_i \in \mathcal{X},\, u_i \in \mathcal{U} \quad \forall i = 0, \dots, N-1 \\ & S x_0 = s \\ & x_N = x_{N-1}, \end{aligned} \tag{6}$$

where $a \in \mathbb{R}^n$ is the cost vector, $S \in \mathbb{R}^{n_s \times n}$ and $s \in \mathbb{R}^{n_s}$ are the initial constraint matrix and vector, with n_s being their size. While the role of a is straightforward, the use of S and s to partially constrain x_0 will become clear in Section IV-B. Let us now prove that the initial state of any locally-optimal solution of (6) is on $\partial\mathcal{V}$.

Lemma 1: Let us consider a locally-optimal state trajectory $\{x_i^*\}_0^N$ computed by solving (6). Let us assume that $P_S a \neq 0$, where $P_S \triangleq (I - S^\dagger S)$ is a null-space projector of S. Then, if N is sufficiently large to allow reaching $\mathcal{S}$ from any viable state, we have:

$$x_0^* \in \partial\mathcal{V}.$$

Moreover, for any sufficiently small value $\epsilon > 0$:

$$\tilde{x}_0 \triangleq x_0^* + \epsilon P_S a \notin \mathcal{V}.$$

Proof: We split this proof in two cases: when $\tilde{x}_0 \notin \mathcal{X}$ and when $\tilde{x}_0 \in \mathcal{X}$. In the first case, $\tilde{x}_0 \notin \mathcal{X}$ implies $\tilde{x}_0 \notin \mathcal{V}$. Moreover, we know by definition of (6) that $x_0^* \in \mathcal{V}$. Since $\tilde{x}_0$ and x_0^* can be arbitrarily close, then we can infer $x_0^* \in \partial\mathcal{V}$. In the second case ($\tilde{x}_0 \in \mathcal{X}$), we can prove this lemma by contradiction. We suppose that $x_0^* \notin \partial\mathcal{V}$, which implies $\tilde{x}_0 \in \mathcal{V}$ for a sufficiently small ϵ, and we show that this leads to the conclusion that x_0^* is not a local optimum. If $x_0^* \notin \partial\mathcal{V}$ then we know that $x_0^* \in \text{int}(\mathcal{V})$. Together with the fact that $\tilde{x}_0 \in \mathcal{X}$, this means that $\tilde{x}_0 \in \mathcal{V}$ for any sufficiently small $\epsilon > 0$. It is easy to verify that $\tilde{x}_0$ satisfies the initial conditions of (6), indeed:

$$S\tilde{x}_0 = Sx_0^* + \epsilon S P_S a = Sx_0^* = s, \tag{7}$$

where we have exploited the fact that $SP_S = 0$. If $\tilde{x}_0$ is viable, by the assumption that N be sufficiently large, it must be possible to satisfy also the terminal conditions of (6), i.e., to reach an equilibrium state. Finally, $\tilde{x}_0$ gives a better cost for (6) than x_0^* because:

$$a^\top \tilde{x}_0 = a^\top x_0^* + \epsilon a^\top P_S a > a^\top x_0^*, \tag{8}$$

where we have exploited the fact that $a^\top P_S a > 0$ because all null-space projectors (as P_S) are positive semi-definite and $P_S a \neq 0$ by assumption. In conclusion, since using $\tilde{x}_0$ as initial state it is possible to satisfy all the constraints of (6), while achieving a better cost, this implies that $\{x_i^*\}_0^N$ be not a local optimum. Therefore, if $\{x_i^*\}_0^N$ is a local optimum, it must hold that $x_0^* \in \partial\mathcal{V}$ and $\tilde{x}_0 \notin \mathcal{V}$. □

Lemma 1 ensures that using problem (6) gives us trajectories that always start from $\partial\mathcal{V}$. However, the remaining N states (from x_1^* to x_N^*) could belong to int($\mathcal{V}$). Ideally, we would like to compute trajectories that are entirely on $\partial\mathcal{V}$. While this is not guaranteed, we argue that often some parts of $\{x_i^*\}_1^N$ are on $\partial\mathcal{V}$, and we provide a simple method to check when this is the case. To this aim, we start by showing that, under certain conditions, a viable state trajectory that starts on $\partial\mathcal{V}$, remains on $\partial\mathcal{V}$ as long as these conditions are met.

Lemma 2: Given a state $x \in \partial\mathcal{V}$, a control $u \in \mathcal{U}$, and a state direction $d \in \mathbb{R}^n$, $||d|| = 1$, such that for any sufficiently small $\epsilon > 0$:

$$\tilde{x} \triangleq x + \epsilon\, d \in \mathcal{X} \setminus \mathcal{V}, \tag{9}$$

then we have that:

$$x^+ = f(x, u) \notin \text{int}(\mathcal{V}). \tag{10}$$

Proof: Since $\tilde{x}$ is in $\mathcal{X}$ but not in $\mathcal{V}$, any state reachable from $\tilde{x}$ cannot be in $\mathcal{V}$; therefore we can write:

$$\begin{aligned} \tilde{x}^+ &= f(\tilde{x}, u) \\ &= f(x, u) + \epsilon\, \partial_x f(x, u) d + O(\epsilon^2) \\ &= x^+ + \epsilon\, \partial_x f(x, u) d + O(\epsilon^2) \notin \mathcal{V}. \end{aligned} \tag{11}$$

Since ϵ can be arbitrarily close to zero, and the eigenvalues of $\partial_x f(x, u)$ are bounded (Assumption 1), this implies that $\tilde{x}^+$ can be arbitrarily close to x^+. Since $\tilde{x}^+ \notin \mathcal{V}$, we can infer that x^+ can either be outside $\mathcal{V}$, or on its boundary, but not in its interior. □

To better understand assumption (9), let us introduce a Corollary, which is a special case of Lemma 2. This Corollary states that if a trajectory starts on $\partial\mathcal{V}$, it cannot reach int($\mathcal{V}$) before reaching $\partial\mathcal{X}$.

Corollary 1: Given a control $u \in \mathcal{U}$ and a state $x \in \partial\mathcal{V}$ satisfying the following assumption:

$$x \notin \partial\mathcal{X}, \tag{12}$$

then we have that: $x^+ = f(x, u) \notin \text{int}(\mathcal{V})$.

Proof: This corollary is a special case of Lemma 2 because (12) implies (9). Indeed, if $x \in \partial\mathcal{V}$ and $x \notin \partial\mathcal{X}$, then there must exist a direction $d \in \mathbb{R}^n$ in which x can be perturbed with an arbitrarily small magnitude ϵ, so that it leaves $\mathcal{V}$ without leaving $\mathcal{X}$, which is what (9) states. □

Lemma 2 states something similar to Corollary 1, but clarifying that actually reaching $\partial\mathcal{X}$ is necessary but not sufficient to reach int($\mathcal{V}$). The real condition to be met is indeed (9). The next Theorem exploits Lemma 2 to suggest a simple method to verify whether the optimal states $\{x_i^*\}_1^N$, computed by solving (6), are on $\partial\mathcal{V}$.

Theorem 1: Let us consider a locally-optimal state trajectory $\{x_i^*\}_0^N$ computed by solving (6). Let us assume that $P_S a \neq 0$, where $P_S \triangleq (I - S^\dagger S)$ is a null-space projector of S. Let us assume that N is sufficiently large to reach $\mathcal{S}$ from any state in $\mathcal{V}$. Consider the following definition of a perturbed state trajectory:

$$\begin{aligned} \tilde{x}_0 &= x_0^* + \epsilon P_S a, \\ \tilde{x}_{i+1} &= f(\tilde{x}_i, u_i^*). \end{aligned} \tag{13}$$

Let us assume that for any sufficiently small $\epsilon > 0$ we have:

$$\tilde{x}_i \in \mathcal{X} \qquad i = 0, \dots, k-1, \tag{14}$$

for a certain time step $k \in [0, N]$. Then we have:

$$x_i^* \in \partial\mathcal{V} \qquad i = 0, \dots, k. \tag{15}$$

Proof: The key idea of this proof is to iteratively apply Lemma 2 to show that $x_{i+1}^* \in \partial\mathcal{V}$, starting from the knowledge that $x_i^* \in \partial\mathcal{V}$, $u_i^* \in \mathcal{U}$, and $\tilde{x}_i \in \mathcal{X} \setminus \mathcal{V}$. We initialize the proof by exploiting Lemma 1, which states that $x_0^* \in \partial\mathcal{V}$ and $\tilde{x}_0 \notin \mathcal{V}$.

Considering also assumption (14) and the obvious fact that $u_i^* \in \mathcal{U}, \forall\, i$, we have all the conditions to apply Lemma 2 for $i = 0$. Lemma 2 tells us only that $x_{i+1}^* \notin \text{int}(\mathcal{V})$. However, since all the optimal states must be viable by definition of (6), we can infer $x_{i+1}^* \in \partial\mathcal{V}$. To iterate the application of Lemma 2 we need to show that $\tilde{x}_{i+1} \in \mathcal{X} \setminus \mathcal{V}$. Assumption (14) ensures that $\tilde{x}_{i+1} \in \mathcal{X}$. The fact that $\tilde{x}_{i+1} \notin \mathcal{V}$ is instead a consequence of $\tilde{x}_i \in \mathcal{X} \setminus \mathcal{V}$ because, by definition of $\mathcal{V}$, we cannot reach $\mathcal{V}$ from the outside without violating a constraint. □

Theorem 1 provides us with the theoretical foundations to design an iterative algorithm for numerically approximating $\mathcal{V}$ in Section V. However, before we do that, the next section analyzes an interesting property of viability kernels of robotic manipulators, which we exploit to customize our algorithm.

IV. Viability for Robot Manipulators

Let us introduce the dynamics of a robot manipulator with n_j DOFs, using an unconventional form for the velocity term [23]:

$$M(q)\ddot{q} + \dot{q}^\top C(q)\dot{q} + g(q) = u, \tag{16}$$

where $q, \dot{q}, \ddot{q} \in \mathbb{R}^{n_j}$ are the joint positions, velocities, and accelerations, $M \in \mathbb{R}^{n_j \times n_j}$ is the positive-definite mass matrix, $C(q) \in \mathbb{R}^{n_j \times n_j \times n_j}$ is the 3D tensor accounting for Coriolis and centrifugal effects, and $g(q) \in \mathbb{R}^{n_j}$ are the gravity torques. We assume that q, $\dot{q}$, and u are bounded:

$$\begin{aligned} q \in \mathcal{Q} &\triangleq \{q \in \mathbb{R}^{n_j} | q^{\min} \leq q \leq q^{\max}\}, \\ \dot{q} \in \dot{\mathcal{Q}} &\triangleq \{\dot{q} \in \mathbb{R}^{n_j} | \dot{q}^{\min} \leq \dot{q} \leq \dot{q}^{\max}\}, \\ u \in \mathcal{U} &\triangleq \{u \in \mathbb{R}^{n_j} | u^{\min} \leq u \leq u^{\max}\}, \end{aligned} \tag{17}$$

where we assume that $\dot{q}^{\min} < 0$ and $\dot{q}^{\max} > 0$.

Assumption 3: Let us assume that the robot is sufficiently strong to compensate for gravity in any configuration:

$$g(q) \in \mathcal{U} \quad \forall\, q \in \mathcal{Q}. \tag{18}$$

In the following lemma we show that, for this class of systems, $\mathcal{V}$ is *star-convex* with respect to the joint velocities. In other words, if a state $(q, \dot{q}) \in \mathcal{V}$, then all states $(q, \alpha\, \dot{q}) \in \mathcal{V}$ for $\alpha \in [0, 1]$.

Lemma 3: Let us consider a manipulator with dynamics (16) and constraints (17). Under Assumption 3, its viability kernel is starred with respect to the joint velocities.

Proof: If a state $(q_0, \dot{q}_0) \in \mathcal{V}$, it means there exists an infinite-time feasible trajectory starting with that state: $(q(t), \dot{q}(t)) \in \mathcal{X}$, $\forall\, t \geq 0$, with $q(0) = q_0$ and $\dot{q}(0) = \dot{q}_0$. We now prove that all the states $(q_0, \alpha\dot{q}_0)$ are viable $\forall\, \alpha \in [0, 1]$ by showing that the time-scaled trajectory $\tilde{q}(t) \triangleq q(\alpha t)$ is feasible. To prove this, we exploit the time-scaling property of manipulator trajectories [23]. The time-scaled trajectory trivially satisfies the joint position and velocity limits, so we only need to prove that it also satisfies the control constraints. The time-scaled joint velocities and accelerations are:

$$\dot{\tilde{q}}(t) = \alpha\dot{q}(\alpha t), \quad \ddot{\tilde{q}}(t) = \alpha^2\ddot{q}(\alpha t). \tag{19}$$

Substituting these expressions in the dynamics (16) we get:

$$\alpha^2(M(q)\ddot{q} + \dot{q}^\top C(q)\dot{q}) + g(q) = \tilde{u}(\alpha), \tag{20}$$

where we expressed the control inputs $\tilde{u}$ as a function of α. Since the original trajectory $q(t)$ is feasible by assumption, we know that $\tilde{u}(1) \in \mathcal{U}$. Moreover, by assumption (18), we know that $\tilde{u}(0) \in \mathcal{U}$. Finally, by convexity of $\mathcal{U}$, we can infer that $\tilde{u}(\alpha) \in \mathcal{U}, \forall\, \alpha \in [0, 1]$, proving the feasibility of the time-scaled trajectory and the viability of $(q_0, \alpha\dot{q}_0)$. □

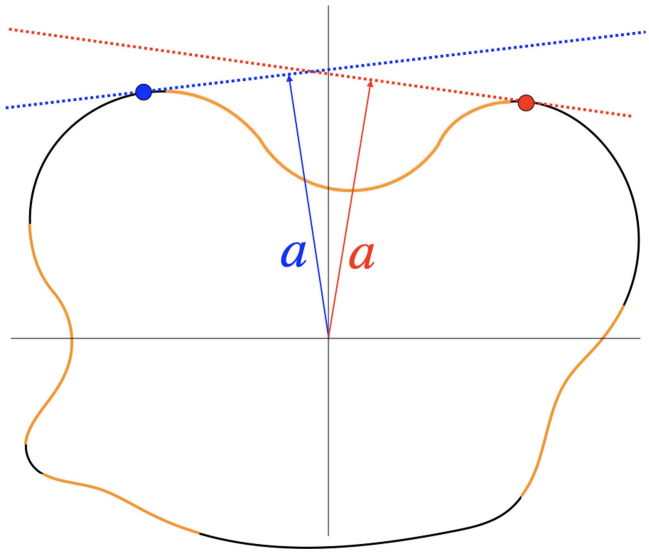

Fig. 1. Star-convex set, with two possible examples of choices of a. The orange parts of the set boundary cannot be discovered by any choice of a.

A. Star-Convex Viability Set Representation

In general, $\mathcal{V}$ can be encoded with a non-parametric classifier, such as a feedforward NN $\phi(x) : \mathbb{R}^n \to \mathbb{R}$, which takes as input a state x and gives as output a binary label (viable, unviable). To train such a classifier, both positive (viable) and negative (unviable) examples are needed. However, so far we have focused on computing viable states on $\partial\mathcal{V}$, which are therefore all positive examples. We could use the perturbed states $\tilde{x}_i(\epsilon)$ (described in Theorem 1) as negative examples, but choosing the proper value of ϵ could be hard. A too small value could lead to positive and negative samples that are too close, making the training of the classifier extremely difficult. On the other hand, a too large value could lead to poor classification accuracy. To avoid these issues, we suggest to exploit that our samples are on $\partial\mathcal{V}$, and that $\mathcal{V}$ is star-convex (Lemma 3) to encode $\mathcal{V}$ differently.

Rather than using a classifier, we could encode $\mathcal{V}$ with a function $\phi(q, d) : \mathbb{R}^{n_j} \times \mathbb{R}^{n_j} \to \mathbb{R}$ that takes as inputs the joint positions q, the joint velocity direction d (with $||d|| = 1$), and computes the maximum viable joint velocity norm. In other words, if $\gamma = \phi(q, d)$, then $(q, \gamma d) \in \partial\mathcal{V}$. With this representation of $\mathcal{V}$, we have transformed the classification problem into a regression problem and we no longer need unviable states to learn $\mathcal{V}$.

B. Uniform Data Distribution

Solving instances of OCP (6) we can compute viable trajectories that are guaranteed to start from $\partial\mathcal{V}$. However, to use these trajectories to learn $\mathcal{V}$, they need to cover its surface as uniformly as possible. This could be hard if $\mathcal{V}$ is non-convex (which is in general the case), since just maximising $a^\top x_0$ for uniformly random directions a would not ensure a complete coverage of $\partial\mathcal{V}$ (e.g., see Fig. 1). In this case, the resulting initial state distribution would depend on the shape of the set

and it could result in an accumulation of data on sharper areas of the set boundary and absence of data in other areas.

Ensuring a uniform coverage of $\partial\mathcal{V}$ does not seem possible without knowing its shape. However, our main concern is to ensure a uniform coverage of the input space of our set representation $\phi(\cdot)$, which is the space of joint angles and joint velocity directions. This is achieved by simply fixing q_0 and the direction of $\dot{q}_0$ to uniformly random values q^{init} and d, while maximizing $||\dot{q}_0||$. This is obtained by choosing:

$$a = \begin{bmatrix} 0 \\ d \end{bmatrix}, \quad S = \begin{bmatrix} I & 0 \\ 0 & I - dd^\top \end{bmatrix}, \quad s = \begin{bmatrix} q^{init} \\ 0 \end{bmatrix}, \tag{21}$$

which leads to a cost that is orthogonal to S, ensuring the satisfaction of the assumption of Lemma 1 ($P_S a \neq 0$).

This choice ensures a uniform distribution of the initial states. However, our method also exploits other states of the optimal trajectories to train the NN, whose distribution depends on the system dynamics. Using the strategy described above, we have observed an accumulation of data at lower velocities, where the trajectories converge to satisfy the terminal constraint. Empirically, we have observed that initializing one joint position at one of its bounds leads to a better coverage of $\partial\mathcal{V}$, because it allows for longer extreme trajectories. The other joint positions are still uniformly randomized, as the initial joint velocity direction. Finally, to avoid trivial instances of the OCP, we ensure that the initial velocity direction of the joint that starts at its bound points away from it (e.g., if $q_0[i] = q_0^{\max}[i]$ then $\dot{q}_0[i] < 0$).

V. Algorithmic Implementation

This section presents the implementation of our algorithm, summarized by the pseudo-code in Algorithms 1 and 2.

Our approach is to generate trajectories that are, at least partially, on $\partial\mathcal{V}$. OCP (6) returns indeed a trajectory that, even if only locally-optimal, is guaranteed to start from $\partial\mathcal{V}$. However, this is true only if the horizon N is sufficiently long. To ensure this is the case, we solve (6) with an increasing value for N, starting from a (reasonable) initial guess, until $a^\top x_0^*$ converges. Algorithm 1 describes this procedure.

To well approximate $\mathcal{V}$ using these states we use the data generation approach described in Section IV-B (lines 3–6 of Algorithm 2).

After solving OCP (6) (line 7 of Algorithm 2), we must check which optimal states belong to $\partial\mathcal{V}$ and can therefore be added to the dataset $\mathcal{D}$. By Lemma 1, we know that $x_0^* \in \partial\mathcal{V}$, and, therefore, we can add it to $\mathcal{D}$ (line 8). Moreover, we know that all viable states on $\partial\mathcal{X}$ are also on $\partial\mathcal{V}$ (simply because $\mathcal{V} \subseteq \mathcal{X}$), so we could add them to $\mathcal{D}$. However, because $\mathcal{V}$ is starred with respect to $\dot{q}$ (see Section IV-A), we only add the states on $\partial\dot{\mathcal{Q}}$ (line 9). At position limits there could be multiple states on $\partial\mathcal{V}$ with the same joint position and velocity direction, but different velocity norms, therefore these states would be conflicting in our starred-set representation and should be discarded. To check if other states are on $\partial\mathcal{V}$, we exploit Theorem 1. We compute the perturbed states $\tilde{x}_i$ as long as they belong to $\mathcal{X}$ (lines 10–13), and we add the associated optimal states to $\mathcal{D}$ (lines 14–15), discarding the states on $\partial\mathcal{X}$ because they have already been considered (line 9).

If the perturbed trajectory leaves $\mathcal{X}$ at $\tilde{x}_{j-1}$, the associated optimal state x_{j-1}^* must be on $\partial\mathcal{X}$ and, therefore, Lemma 2 tells us that the rest of the state trajectory could belong to int($\mathcal{V}$). To check if this is the case, we exploit again Lemma 3, which tells us that the states on $\partial\mathcal{V}$, except for those at joint position limits, have maximum velocity norm for that position and velocity direction. So, when the optimal trajectory leaves $\partial\mathcal{X}$, say at $x_j^* \notin \partial\mathcal{X}$ (line 18), we solve another OCP (6), fixing the initial position and velocity direction to those of x_j^* (lines 19–22). If this OCP returns the same initial velocity norm of x_j^*, then this proves that $x_j^* \in \partial\mathcal{V}$. If instead the OCP returns a higher initial velocity norm, it means that x_j^* was in int($\mathcal{V}$), but it gives anyway a new trajectory starting from $\partial\mathcal{V}$ that can be used in place of the previous one (lines 23–26). In both cases, x_j^* can be added to $\mathcal{D}$ (line 27) and the whole process can continue.

Algorithm 1: Viability-Boundary Optimal Control (VBOC).

Require: Constraint sets $\mathcal{X}$ and $\mathcal{U}$, Dynamics $f(\cdot,\cdot)$, Number of DOFs n_j, Time horizon N_{start}, Time horizon increment n, OCP (6), Initial joint positions q^{init}, Initial joint velocity direction d

1: $a, s, S \leftarrow (21)$
2: $N, \gamma \leftarrow N_{start}, 0$
3: **repeat**
4: $\quad \{x_i^*\}_0^N, \{u_i^*\}_0^{N-1} \leftarrow$ OCP($\mathcal{X}$,$\mathcal{U}$, f, N, a, S, s)
5: $\quad \gamma_{previous}, N \leftarrow \gamma, N + n$
6: $\quad \gamma \leftarrow a^\top x_0^*$
7: **until** $\gamma > \gamma_{previous}$
8: **return** $\{x_i^*\}_0^N, \{u_i^*\}_0^{N-1}, N, a$

VI. Results

To study the performance of our algorithm (VBOC) we test it with 2, 4 and 6-dimensional systems. We compare VBOC with two state-of-the-art algorithms, focusing the comparison on the data-generation part, which is our main contribution. The chosen algorithms are: i) the approach presented in Section II-D relying on an informative-based Active Learning (AL) algorithm [22], and ii) a Hamilton-Jacoby Reachability (HJR) algorithm [13]. HJR is an approximate dynamic programming algorithm that computes the solution of the HJI PDE through recursive regression (since we are interested in infinite-time backward reachability, we discard the time dependency).

We evaluated the accuracy of the results by generating a test set using only the initial states obtained by calling Algorithm 1 with fully random initial position and velocity direction, to obtain well distributed samples on $\partial\mathcal{V}$ (using the whole state trajectories would result in a higher density of samples at low velocities). On this set of N points, we measured the Root Mean Squared Error (RMSE), defined as:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1}\left(||\dot{q}_i|| - \phi\left(q_i, \frac{\dot{q}_i}{||\dot{q}_i||}\right)\right)^2}, \tag{22}$$

where $\phi(\cdot,\cdot)$ is the NN trained by VBOC. For AL and HJR, the trained NN is instead a classifier. Therefore, to measure the RMSE, we numerically identify (via binary search) the classifier boundary for the given joint positions q_i and velocity direction $\dot{q}_i/||\dot{q}_i||$.

In the tests we have run VBOC with $N_{start} = 100$, $\epsilon = 10^{-2}$ and solver tolerances equal to 10^{-3}.

Algorithm 2: Compute States on $\partial\mathcal{V}$.

Require: Constraint sets $\mathcal{Q}$ and $\dot{\mathcal{Q}}$, Number of DOFs n_j Number of trajectories K, Perturbation parameter ϵ, Time horizon N_{guess}

$\mathcal{D} \leftarrow []$
for $k = 0 \rightarrow K$ **do**
  $q \leftarrow \text{RANDOMUNIFORM}(q^{\min}, q^{\max})$
  $i \leftarrow \text{RANDOMINTEGER}(0, n_j)$
  $q[i] \leftarrow \text{RANDOMCHOICE}([q^{\min}[i], q^{\max}[i]])$
  $d \leftarrow \text{RANDOMVELOCITYDIRECTION}(i, q)$
  $\{x_i^*\}_0^N, \{u_i^*\}_0^{N-1}, N, a \leftarrow \text{VBOC}(q, d, N_{guess})$
  insert x_0^* **in** $\mathcal{D}$
  for $l = 1 \rightarrow N$ **do**
    if $\dot{q}_l^* \in \partial\dot{\mathcal{Q}}$ **insert** x_l^* **in** $\mathcal{D}$
  $\tilde{x}_0 \leftarrow x_0^* + \epsilon a$
  for $j = 1 \rightarrow N$ **do**
    if $\tilde{x}_{j-1} \in \mathcal{X}$ **then**
      $\tilde{x}_j \leftarrow f(\tilde{x}_{j-1}, u_{j-1}^*)$
      if $x_j^* \notin \partial\mathcal{X}$ **insert** x_j^* **in** $\mathcal{D}$
    else
      $\tilde{x}_j \leftarrow \tilde{x}_{j-1}$
      if $x_j^* \notin \partial\mathcal{X}$ **then**
        $\gamma \leftarrow a^\top x_j^*$
        $d \leftarrow \text{VELOCITYDIRECTION}(x_j^*)$
        $q \leftarrow \text{JOINTPOSITIONS}(x_j^*)$
        $\{x_i^*\}_j^N, \{u_i^*\}_j^{N-1}, \sim, a \leftarrow \text{VBOC}(q, d, N - j)$
        $\gamma_{new} \leftarrow a^\top x_j^*$
        if $\gamma_{new} > \gamma$ **then**
          for $l = j + 1 \rightarrow N$ **do**
            if $\dot{q}_l^* \in \partial\dot{\mathcal{Q}}$ **insert** x_l^* **in** $\mathcal{D}$ **then**
        insert x_j^* **in** $\mathcal{D}$
        $\tilde{x}_j \leftarrow x_j^* + \epsilon a$
return $\mathcal{D}$

We have used fully-connected NNs composed of 3 layers with ReLU activation functions. All algorithms are implemented[1] in Python, using ACADOS [24] for solving the OCPs and the PyTorch [25] implementation of Adam [26] for the NNs training. The tests are performed on a computer with 32 AMD Ryzen9 5950x processors and a GeForce RTX 3060 GPU. The OCPs are solved in parallel on 30 cores and the NNs training is performed with CUDA on the GPU.

A. Tests on a 2D System

The tested system is a simple pendulum, a model with a single swinging link connected to a fixed base through a revolute joint. The system has a 2-dimensional (2D) state space $x = [q\ \dot{q}]^\top$ and a 1-dimensional (1D) control input u. The joint positions, velocities and input constraints are in the form (17) and are set to $\pi \pm \pi/4$ rad, ± 10 rad/s, and ± 3 Nm, respectively. With a 2D system, the computation of $\mathcal{V}$ is actually simpler than explained in Algorithm 2 since its boundary does not require sampling to be explored. It is sufficient to generate two trajectories with initial positions fixed at the two extremes $q^{\min}/q^{\max}$. The used NN has 100 neurons in the hidden layer. The algorithm converged in 13 s and the resulting $\mathcal{V}$ approximation and the training data are shown in Fig. 2.

[1]Our code is available at github.com/idra-lab/VBOC.

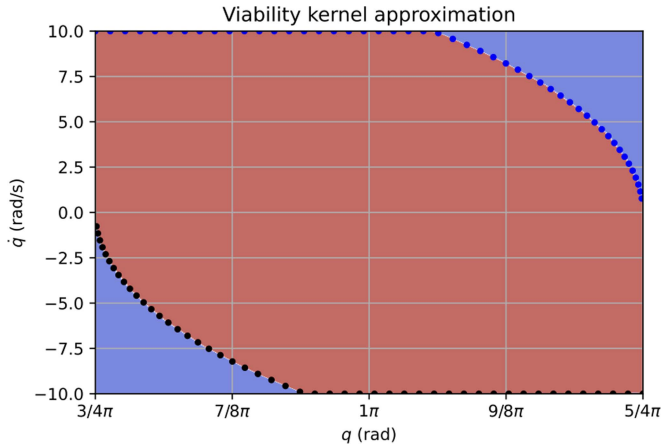

Fig. 2. Viability kernel for the single pendulum. The background color represents the set learned using VBOC. The black and blue dots represent the training data from the two generated trajectories. The axis limits correspond to the joint position and velocity limits.

TABLE I
RMSE COMPARISON FOR THE 2D SYSTEM

	VBOC (ours)	AL	HJR
TIME (S)	13	22	117
RMSE TESTING (RAD/S)	0.0206	0.0477	0.3899

To highlight the improvement with respect to the other state-of-the-art approaches, we compare the computational time and RMSE obtained using the same solver and NN complexity. AL and HJR have been executed on a grid with 100^2 samples. Table I shows the results. VBOC results to be faster and more accurate because the other approaches, having to sample the state space and train the NN multiple times due to their iterative structure, require more time to obtain good approximations. HJR performed worse than the others because, at each iteration, it relies on the NN trained at the previous iteration; this leads to an accumulation of the approximation error.

B. Tests on a 4D System

The tested system is now a double pendulum, an open kinematic chain with two swinging links and two planar revolute joints. The system has a 4D state space $x = [q_1\ q_2\ \dot{q}_1\ \dot{q}_2]^\top$ and a 2D control input $u = [\tau_1\ \tau_2]^\top$. The state constraints of each joint are the same of the previous test, the input constraints are now ± 10 Nm.

In these tests we decided to compare the RMSE evolutions for the three algorithms while learning. VBOC, as presented in Algorithm 2, is not incremental, but we can easily make it incremental by alternating between data generation and training. At each iteration we computed a batch of $K = 1000$ data points. The AL algorithm is executed it on a grid with 60^4 samples and batches of 1000 points. For HJR, to converge in a reasonable amount of time (58 minutes), we had to use a smaller number of samples: 20^4. For VBOC and AL we have used NNs with 300 neurons in the hidden layer. For HJR instead, since the NN is

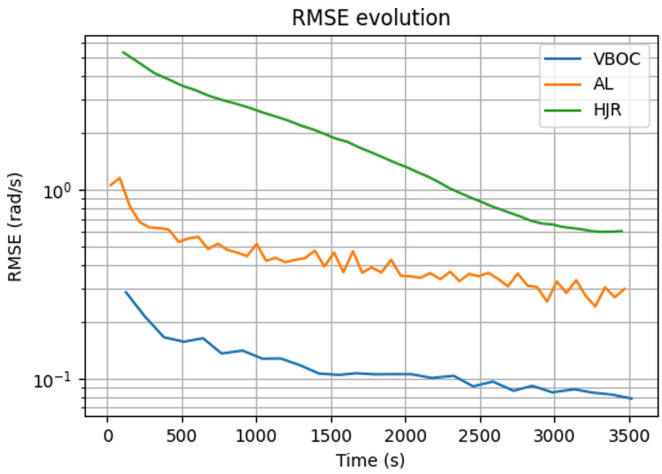

Fig. 3. Comparison between the RMSE evolution for the 4D system.

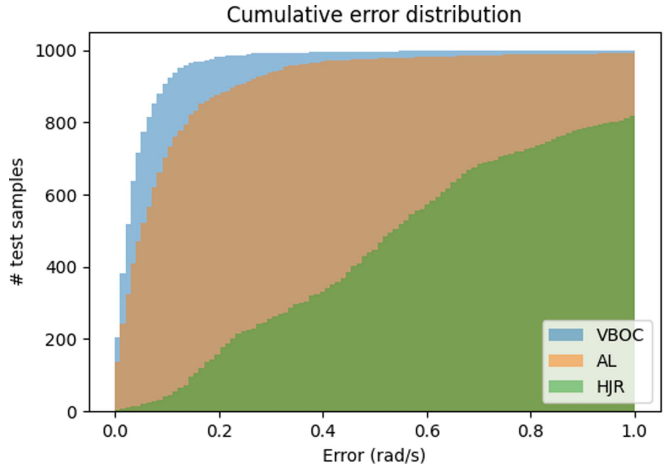

Fig. 4. Cumulative error distribution for the 4D system at then end of the training. This plot shows how many test samples (y axis) obtained a prediction error (x axis) below a certain value.

TABLE II
RMSE COMPARISON FOR THE 4D SYSTEM

	VBOC (ours)	AL	HJR
RMSE TESTING (RAD/S)	0.0782	0.2693	0.6002
RMSE TRAINING (RAD/S)	0.0636	–	–

here used inside the OCPs, we observed that it performed better using only 100 neurons due to the reduced complexity of the OCPs.

Fig. 3 shows the RMSE evolutions over time, where we can see that VBOC reaches higher accuracies much faster than the other algorithms. At each iteration, HJR has to solve an OCP for each point and to train an accurate NN. Even if the OCPs have a shorter (1-step) horizon, the higher number of OCPs and the training of the intermediate NNs limit the number of samples that can be used and, consequently, the final accuracy. AL and VBOC point instead at minimizing the number of solved OCPs, which allows them to be more efficient. Table II reports the final RMSE error for the three algorithms, which is 3.4 times larger for AL than for VBOC. Moreover, Table II also reports the RMSE of VBOC on the training set, which is only slightly better than on the test set, highlighting a good generalization capability of the trained NN. Finally, looking at the cumulative error distribution in Fig. 4 we can see that VBOC not only resulted in a smaller average error (RMSE), but also in a lower number of errors above any given threshold.

C. Tests on a 6D System

To test the scalability of VBOC, we applied it also to a triple pendulum, which has a 6D state space $x = [q_1\ q_2\ q_3\ \dot{q}_1\ \dot{q}_2\ \dot{q}_3]^\top$ and a 3D control input $u = [\tau_1\ \tau_2\ \tau_3]^\top$. The state and input constraints are the same as the previous test. For this system we compare only VBOC and AL, since the curse of dimensionality of HJR resulted to be already too relevant. Indeed, HJR converged with an extremely high RMSE (4.0 rd/s) because we had to execute it on a grid of only 12^6 points to make it converge within 6 hours.

Table III shows that the final RMSE obtained by VBOC is 2.2 times smaller than the one obtained with AL. Fig. 5 shows

TABLE III
RMSE COMPARISON FOR THE 6D SYSTEM

	VBOC (ours)	AL
RMSE TESTING (RAD/S)	0.1752	0.3693
RMSE TRAINING (RAD/S)	0.0863	–

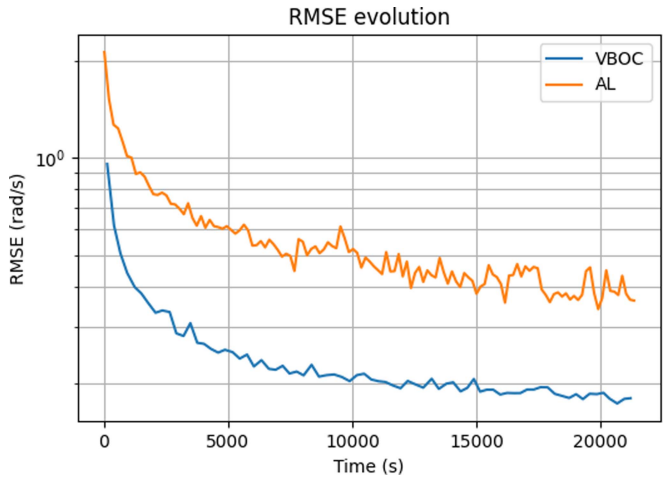

Fig. 5. Comparison between the RMSE evolution for the 6D system.

that the RMSE achieved by AL after 6 hours is comparable to that achieved by VBOC after less than 30 minutes. Even though the RMSE of VBOC on the training set is comparable to the one obtained for the 4D system, errors on the test set are larger, highlighting a lack of generalization. Smaller errors could be achieved by letting the algorithms run for more iterations, and/or using larger NN structures.

VII. CONCLUSION

This letter presented a new algorithm (VBOC) for the approximation of the viability kernel of robot manipulators. Differently from state-of-the-art approaches, VBOC computes directly

states on the boundary of the set, leading to more accurate results in a data-efficient manner. The set boundary has indeed a smaller dimension than the set itself, so VBOC scales more favourably than algorithms that explore the entire state space. VBOC is theoretically guaranteed to provide data on the boundary of the set, requiring only local optimality of the OCP solutions. Moreover, contrary to many state-of-the-art methods, VBOC does not need to rely on the ability of the OCP solver to correctly detect unfeasible problems, which makes it robust to numerical errors. Additionally, since the trained NN introduces some approximation errors due to its intrinsic inability to exactly represent the set, the OCPs complexity can be reduced by relaxing the solver tolerances to be only slightly more accurate than the expected error on the set approximation.

Despite all of this, many issues still remain open. For instance, scalability is still a major concern, since the algorithmic complexity still scales exponentially. To address this, we could use customized NN structures that embed our prior knowledge on the shape of $\mathcal{V}$ (e.g., we know that the maximum viable velocity is null when a joint is at its bound). Moreover, even if our tests have focused on joint-space constraints, we plan to extend VBOC to Cartesian-space constraints, e.g., for obstacle avoidance. Another interesting challenge is the use of the learned sets as terminal constraints in MPC (or safety filters in Reinforcement Learning). Since these sets are *approximations* of $\mathcal{V}$, they are only *approximately* control invariant, so recursive feasibility cannot be guaranteed in general. Therefore, we plan to investigate algorithmic approaches to use these sets, while maintaining strong guarantees of safety. Finally, while the theory in Section III holds for any differentiable system, our algorithm is specifically designed for star-convex sets; its extension to more generic cases is currently being investigated.

REFERENCES

[1] F. Blanchini, "Set invariance in control," *Automatica*, vol. 35, pp. 1747–1767, 1999.

[2] A. D. Prete, "Joint position and velocity bounds in discrete-time acceleration/torque control of robot manipulators," *IEEE Robot. Automat. Lett.*, vol. 3, no. 1, pp. 281–288, Jan. 2018.

[3] P. Saint-Pierre, "Approximation of the viability kernel," *Appl. Math. Optim.*, vol. 29, pp. 187–209, 1994.

[4] J.-P. Aubin, *Viability Theory*, 1st ed., C. I. Byrnes, Ed. Berlin, Germany: Springer, 1991.

[5] F. Rußwurm, W. Esterhuizen, K. Worthmann, and S. Streif, "On MPC without terminal conditions for dynamic non-holonomic robots," *IFAC-PapersOnLine*, vol. 54, no. 6, pp. 133–138, 2021.

[6] P.-A. Coquelin, S. Martin, and R. Munos, "A dynamic programming approach to viability problems," in *Proc. IEEE Int. Symp. Approx. Dynamic Program. Reinforcement Learn.*, 2007, pp. 178–184.

[7] N. Bonneuil, "Computing the viability kernel in large state dimension," *J. Math. Anal. Appl.*, vol. 323, pp. 1444–1454, 2006.

[8] I. M. Mitchell, "Comparing forward and backward reachability as tools for safety analysis," in *Proc. Int. Conf. Hybrid Systems: Computation Control*, 2007, pp. 428–443.

[9] D. Monnet, J. Ninin, and L. Jaulin, "Computing an inner and an outer approximation of the viability kernel," *Reliable Comput.*, vol. 22, 2016.

[10] J. M. Bravo, D. Limon, T. Alamo, and E. F. Camacho, "On the computation of invariant sets for constrained nonlinear systems: An interval arithmetic approach," in *Proc. Eur. Control Conf.*, 2003, pp. 288–293.

[11] E. Zanolli and A. Del Prete, "Robust satisfaction of joint position and velocity bounds in discrete-time acceleration control of robot manipulators," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2023.

[12] F. Jiang, G. Chou, M. Chen, and C. J. Tomlin, "Using neural networks to compute approximate and guaranteed feasible Hamilton-Jacobi-Bellman PDE solutions," 2016, *arXiv:1611.03158*.

[13] V. Rubies-Royo and C. Tomlin, "Recursive regression with neural networks: Approximating the HJI PDE solution," in *Proc. 5th Int. Conf. Learn. Representations*, 2017.

[14] B. Djeridane and J. Lygeros, "Neural approximation of PDE solutions: An application to reachability computations," in *Proc. IEEE Conf. Decis. Control*, 2006, pp. 3034–3039.

[15] K.-C. Hsu, V. Rubies-Royo, C. J. Tomlin, and J. F. Fisac, "Safety and liveness guarantees through reach-avoid reinforcement learning," 2021, *arXiv:2112.12288*.

[16] Y. Zhou, D. Li, Y. Xi, and Y. Xu, "Data-driven approximation for feasible regions in nonlinear model predictive control," 2020, *arXiv:2012.03428*.

[17] L. Chapel and G. Deffuant, "SVM viability controller active learning," in *Proc. Kernel Machines Reinforcement Learn. Workshop*, 2007, pp. 193–200.

[18] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," *Annu. Rev. Control Robot. Auton. Syst.*, vol. 3, pp. 269–296, 2020.

[19] K. P. Wabersich and M. N. Zeilinger, "Linear model predictive safety certification for learning-based control," in *Proc. IEEE Conf. Decis. Control*, 2018, pp. 7130–7135.

[20] B. Djeridane and J. Lygeros, "Approximate viability using quasi-random samples and a neural network classifier," *IFAC Proc. Volumes*, vol. 41, pp. 14342–14347, 2008.

[21] A. Chakrabarty, A. Raghunathan, S. D. Cairano, and C. Danielson, "Data-driven estimation of backward reachable and invariant sets for unmodeled systems via active learning," in *Proc. IEEE Conf. Decis. Control*, 2019, pp. 372–377.

[22] A. Chakrabarty, C. Danielson, S. D. Cairano, and A. Raghunathan, "Active learning for estimating reachable sets for systems with unknown dynamics," *IEEE Trans. Cybern.*, vol. 52, no. 4, pp. 2531–2542, Apr. 2022.

[23] J. M. Hollerbach, "Dynamic scaling of manipulator trajectories," in *Proc. IEEE Amer. Control Conf.*, 1983, pp. 752–756.

[24] R. Verschueren et al., "Acados—A modular open-source framework for fast embedded optimal control," *Math. Program. Computation*, vol. 14, pp. 147–183, 2019.

[25] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Conf. Neural Inf. Process. Syst.*, 2019.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, 2014, pp. 14–17.

Open Access funding provided by 'Universit? degli Studi di Trento' within the CRUI CARE Agreement