On exploration requirements for learning safety constraints

Pierre-François Massiani[1, 2] MASSIANI@DSME.RWTH-AACHEN.DE
Steve Heim[1] HEIM.STEVE@GMAIL.COM
Sebastian Trimpe[1, 2] TRIMPE@DSME.RWTH-AACHEN.DE
[1] *Intelligent Control Systems Group, Max Planck Institute for Intelligent Systems, Stuttgart, Germany*
[2] *Data Science in Mechanical Engineering, RWTH Aachen University, Aachen, Germany*

Editors: A. Jadbabaie, J. Lygeros, G. J. Pappas, P. A. Parrilo, B. Recht, C. J. Tomlin, M. N. Zeilinger

Abstract

Enforcing safety for dynamical systems is challenging, since it requires constraint satisfaction along trajectory predictions. Equivalent control constraints can be computed in the form of sets that enforce positive invariance, and can thus guarantee safety in feedback controllers without predictions. However, these constraints are cumbersome to compute from models, and it is not yet well established how to infer constraints from data. In this paper, we shed light on the key objects involved in learning control constraints from data in a model-free setting. In particular, we discuss the family of constraints that enforce safety in the context of a nominal control policy, and expose that these constraints do not need to be accurate everywhere. They only need to correctly exclude a subset of the state-actions that would cause failure, which we call the critical set.

Keywords: Safety, Constraints, Viability, Positive Invariance

1. Introduction

Safety is an important and challenging consideration in controller design, especially when deploying systems whose dynamics or environment is difficult to model or otherwise uncertain. For such systems, data-driven or 'learning' based controller design, such as reinforcement learning, has recently had a lot of success (Recht, 2019; Chatzilygeroudis et al., 2020; Ibarz et al., 2021).However, most work has focused on learning optimal controllers, while learning safety is not as well understood.

There are various formalizations to ensure safety (Garcıa and Fernández, 2015). We will consider safety as state constraint satisfaction: there is a *failure set* of states that the system must never visit. For dynamical systems, however, it is insufficient to simply constrain the controller to stay in the complement of this set: there may be states that are not in the failure set, but from which failure can no longer be avoided within finite time. Instead, a prediction of unknown horizon length needs to be made to guarantee that the trajectory never passes through the failure set. Model-predictive control (MPC) provides a natural description of this problem by predicting these trajectories at run-time. This approach has been successful in many domains, but also has a few drawbacks: in particular, it requires a sufficiently accurate model and substantial computational resources, such that predictions and optimization can be run fast enough during run-time (Hewing et al., 2020; Nubert et al., 2020).

We will focus on another route, which provides control constraints that can be applied to a feedback controller, and relies on (pre-)computing controlled invariant sets: a set of states for which there exists a control policy that keeps the system in this set. These sets can often be computed

using tools from the fields of viability theory (Aubin et al., 2011), back-reachability (Bansal et al., 2017), and control barrier functions (Ames et al., 2019). However, these tools are also contingent on accurate models and are often too computationally demanding even for offline analysis.

The requirement for predictions (and therefore a model) can be entirely sidestepped by directly learning control constraints from data. The objects of interest here, *control constraints*, are sets in state-action space that can be used to constrain a control policy such that the system never leaves the maximal control invariant set, also called the viability kernel. Recent work in this direction has provided ways to learn the largest such control constraint (Heim et al., 2019; Fisac et al., 2019). While they successfully sidestep the need for a model, learning the entire maximal control constraint is highly sample-inefficient and often unnecessary.

In this paper, we first systematically identify the family of constraints that can guarantee positive invariance, and therefore safety, in the general setting. We then reformulate the problem of constraining a given nominal policy as a standard optimization problem, which exposes a new object we call the *critical set* of state-action pairs. We prove that the inclusion of the true optimal policy and exclusion of the critical set are sufficient and necessary conditions for a constraint set to enforce safety without resulting in a suboptimal policy. A key insight is that, because the critical set is a subset of all failure-inducing state-actions, it is not necessary to fully explore the state-action space to converge to a safe, optimal policy. We believe this insight will inform the design of learning algorithms, in the context of a nominal policy, in two ways: first, it suggests that safe sets do not always need to be estimated conservatively. Second, it shows that a greedy exploration scheme is sufficient for learning safety. We also provide code to reproduce numerical results in Section 4.

1.1. Related Work

Model predictive control (MPC) is perhaps the most popular control paradigm for guaranteeing constraint satisfaction (Hewing et al., 2020). Since constraints are enforced along an entire trajectory by formulating an optimization problem along a predicted trajectory, its performance hinges on identifying or learning an accurate model. In addition, the problem is often formulated with simplified dynamics and quadratic costs, such that it can be efficiently solved via convex optimization. In contrast, we aim to learn control constraints that can be directly evaluated in a feedback controller. Learning feedback constraints has substantial challenges; however, once learned, it benefits from being agnostic to the cost formulation and bypasses the need for a model. It also has much lower computational requirements during run-time. We see these advantages as particularly relevant in combination with model-free learning algorithms such as deep Q-learning (Xie et al., 2020).

These control constraints can be computed as sets using tools from back-reachability (Fisac et al., 2018; Kaynama et al., 2012) and viability theory (Wieber, 2018; Aubin et al., 2011; Liniger and Lygeros, 2019), or as the superlevel-set of a control barrier function (Ames et al., 2019). Computing safe sets for arbitrary dynamical systems, however, relies on computation-intensive, model-based algorithms, which often limits their use to offline analysis, and/or for low-dimensional systems where accurate models are available. The computational challenge is often addressed by using less accurate, low-dimensional models to obtain set approximations, then bounding the tracking error of the full system (Fridovich-Keil et al., 2018, 2019; Khadiv et al., 2020; Zhou et al., 2020). The model sensitivity challenge is often addressed by refining a set approximation with data, potentially gathered during run-time. Samples can be used to directly improve the dynamics or disturbance models (Fisac et al., 2018), but more recently, there has been interest in completely

bypassing the model and directly updating the control constraint estimate (Shih et al., 2020; Fisac et al., 2019; Heim et al., 2019; Raković et al., 2017; Boffi et al., 2020; Robey et al., 2020). In this case, the computational cost is typically traded for sample-inefficiency. Regardless of the approach, it is common practice to use conservative inner approximations. This is very reasonable, as only a conservative approximation of the set will maintain its guarantees. The key insight of our work is that, when used in the context of a specific controller, this conservativeness is unnecessary. We show that it is sufficient to converge to a family of constraint sets that includes not only (most) conservative approximations, but also many over-approximations, as long as they satisfy specific requirements. This relaxes the requirements on constraint-learning algorithms, and also shows that greedy, on-policy exploration naturally fulfills these requirements.

2. Viability, Positive Invariance, and Control Constraints

We consider a discrete-time system with state $s_k \in \mathcal{S} \subset \mathbb{R}^n$, control input $a_k \in \mathcal{A} \subset \mathbb{R}^m$, and dynamics $s_{k+1} = T(s_k, a_k)$, where $k \in \mathbb{N}$ indicates the time-step. Assumptions on T are introduced in Section 3.1. In addition, we consider a set of failure states $\mathcal{S}_\text{F}$; we assume this to be an absorbing set, e.g., once the system enters this set, it stays there forever and can no longer take actions. We define $\mathcal{Q} = (\mathcal{S} \setminus \mathcal{S}_\text{F}) \times \mathcal{A}$ as the state-action space without the failure set.

Our goal in this section is to define a set $\mathcal{Q}_\text{PI}$ in state-action space that can be used to constrain a policy such that it induces a positive invariant set in state space. By constrain, we mean that the policy must map any state s to an action a such that $(s, a) \in \mathcal{Q}_\text{PI}$ whenever such an action exists. By positive invariance, we mean that all trajectories that start in a given set in state space will remain in this set forever when following a policy constrained by $\mathcal{Q}_\text{PI}$, and therefore stay out of $\mathcal{S}_\text{F}$.

We will start by introducing key concepts related to viability theory and positive invariance, which will allow us to reason about dynamical systems in terms of sets in state-action space without explicitly considering trajectories. These objects will allow us to reformulate the safe control problem as a standard constrained optimization problem in Section 3.

Running example We illustrate these objects on the hovership toy model used in (Heim et al., 2019). This system has a one-dimensional state space $s \in [0, 2]$ and a one-dimensional action space $a \in [0, 0.8]$, and continuous-time dynamics $\dot{s} = a - 0.1 - \tanh(0.75s)$. We treat this as a discrete-time system by applying control actions with a zero-order hold for one second, and integrating the continuous-time dynamics. Figure 1 illustrates all the objects introduced in this section using this running example.

We start by defining the viability kernel $\mathcal{S}_\text{V}$:

Definition 1 (Viability Kernel) *The* viability kernel $\mathcal{S}_V \subset \mathcal{S} \setminus \mathcal{S}_F$ *is the maximal set of states from which there exists an action that keeps the system inside* $\mathcal{S}_V$ *(cf. (Aubin et al., 2011, Chapter 1.1)).*

In other words, all states outside of $\mathcal{S}_\text{V}$ will fail within finite time, regardless of the action chosen.

Next, we define the set of state-action pairs that map into the viability kernel:

Definition 2 (Viable Set) *The* viable set $\mathcal{Q}_V \subset \mathcal{Q}$ *is the maximal set of state-actions* (s, a) *such that* $s_{k+1} = T(s_k, a_k) \in \mathcal{S}_V$*. (cf. (Heim et al., 2019, Definition 2)).*

The viable set $\mathcal{Q}_\text{V}$ can be directly used as a safety constraint: in fact, it is both necessary and sufficient for a safe policy to exclusively map viable states to the viable set. It is necessary, since all

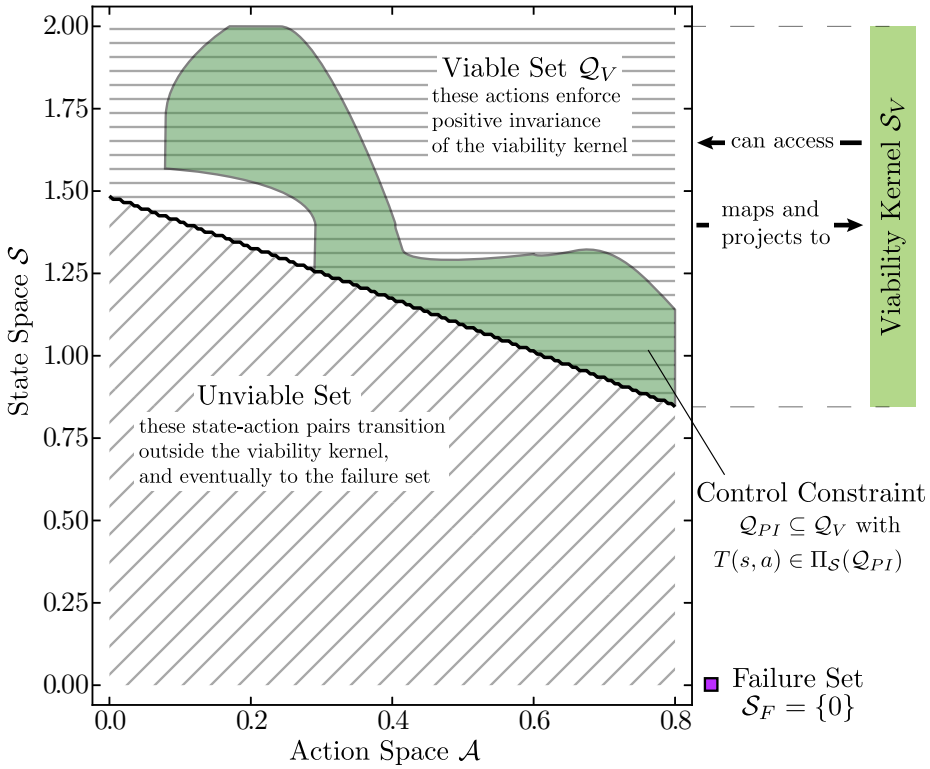

Figure 1: A system's long-term behavior can be captured by the viable set $\mathcal{Q}_{\text{V}}$ (horizontal hatch) and its complement, the unviable set (diagonal hatch). All state-action pairs in the unviable set form trajectories that terminate in a failure set in finite time (purple). State-action pairs in $\mathcal{Q}_{\text{V}}$ transition to the viability kernel $\mathcal{S}_{\text{V}}$ (green, right). Since $\mathcal{S}_{\text{V}}$ is the projection of $\mathcal{Q}_{\text{V}}$, states in $\mathcal{S}_{\text{V}}$ can recursively choose actions in $\mathcal{Q}_{\text{V}}$ to avoid failure forever. The same holds for any control constraint (green, horizontal hatch) that also has $\mathcal{S}_{\text{V}}$ as a projection.

state-action pairs outside $\mathcal{Q}_{\text{V}}$ map to states outside $\mathcal{S}_{\text{V}}$, leading to the failure set in finite time. It is also sufficient, since all of $\mathcal{Q}_{\text{V}}$ maps into $\mathcal{S}_{\text{V}}$, and $\mathcal{Q}_{\text{V}}$ can always be accessed by states in $\mathcal{S}_{\text{V}}$.

This second property is a key observation that holds for any control constraint $\mathcal{Q}_{\text{PI}}$: to enforce positive invariance, the projection of $\mathcal{Q}_{\text{PI}}$ into state space, $\Pi_{\mathcal{S}}(\mathcal{Q}_{\text{PI}}) = \{s, (s,a) \in \mathcal{Q}_{\text{PI}}\}$, must be a cover of the set of states mapped from $\mathcal{Q}_{\text{PI}}$. We can now define arbitrary control constraints $\mathcal{Q}_{\text{PI}}$:

Definition 3 (Control constraint) *We call a* control constraint *a set $\mathcal{Q}_{PI} \subset \mathcal{Q}$ of state-action pairs where, for all $(s,a) \in \mathcal{Q}_{PI}$ we have $T(s,a) \in \Pi_{\mathcal{S}}(\mathcal{Q}_{PI})$.*

Control constraints allow us to consider the state-action space in a static sense, without the need to consider trajectories. This is because they map back into their own projection: given a control constraint $\mathcal{Q}_{\text{PI}}$, the next state-action pair chosen can (and will) be selected from $\mathcal{Q}_{\text{PI}}$.

Remark 4 *If A and B are control constraints, and C is any subset of $\mathcal{Q}$, the following hold:*

1. *$A \cup B$ is a control constraint;*

2. *If* $\Pi_{\mathcal{S}}(A \setminus C) = \Pi_{\mathcal{S}}(A)$*, then* $A \setminus C$ *is a control constraint;*

3. $A \subseteq \mathcal{Q}_V$.

We see that $\mathcal{Q}_{\mathrm{V}}$ is the largest control constraint, but not every subset of $\mathcal{Q}_{\mathrm{V}}$ is a control constraint. Control constraints can be used to guarantee safety for any policy π: by enforcing $(s, \pi(s)) \in \mathcal{Q}_{\mathrm{PI}}$, we have a policy that is safe for any initial conditions starting in the projection of $\mathcal{Q}_{\mathrm{PI}}$. In the next Section we will show that, if we are only interested in constraining a *specific* policy, we can consider a class of sets with less restrictive requirements.

3. Constraints Don't Need to be Accurate Everywhere

We now consider the problem of learning constraints for a given nominal policy π. We will show that, by formulating the control policy as a constrained optimization problem, we can consider a much less restrictive set of constraints $\mathcal{K}$ than control constraints. In addition, this formulation shows that greedily following the nominal policy naturally prioritizes exploring the relevant portion of the state-action space, and off-policy exploration is not necessary.

3.1. Problem Statement

In addition to the setting introduced in Section 2, we consider a given nominal policy π, and a cost function $J : \mathcal{Q} \to \mathbb{R}$. We can then define a controller constrained by a set $\mathcal{K}$ as the solution of the optimization problem $\mathcal{OPT}$:

$$\begin{aligned} \mathcal{OPT}(\mathcal{K}) \ : s \mapsto \ & \underset{a \in \mathcal{A}}{\operatorname{argmin}} \ J(s, a), \\ & \text{s.t.} \quad (s, a) \in \mathcal{K}. \end{aligned} \tag{1}$$

For simplicity of exposition, we only consider deterministic policies, and use the euclidean distance to π as the cost function $J(s, a) = \|a - \pi(s)\|^2$, though other costs can be used instead. We also require that $\mathcal{K}$ is closed to ensure the existence of the argmin.

If we happen to know $\mathcal{Q}_{\mathrm{V}}$, we can directly plug $\mathcal{Q}_{\mathrm{V}}$ into $\mathcal{OPT}$ to find the viable policy that achieves the lowest cost. Note that, to ensure that $\mathcal{OPT}(\mathcal{Q}_{\mathrm{V}})$ has a well-defined solution, we assume that $\mathcal{Q}_{\mathrm{V}}$ is closed: to satisfy this assumption, the system dynamics T must generate a closed viable set[1].

Since $\mathcal{Q}_{\mathrm{V}}$ is generally unknown, our goal is to find the requirements on $\mathcal{K}$, such that it produces the same policy:

$$\text{find } \mathcal{K}, \quad \text{s.t.} \quad \mathcal{OPT}(\mathcal{K}) = \mathcal{OPT}(\mathcal{Q}_{\mathrm{V}}). \tag{2}$$

Notice how the only control constraints that satisfy this property are the ones that contain $\mathcal{OPT}(\mathcal{Q}_{\mathrm{V}})$. However, there are many sets $\mathcal{K}$ that are not control constraints, nor even subsets of $\mathcal{Q}_{\mathrm{V}}$.

3.2. Admissible Constraints $\mathcal{K}$

We have visualized the requirements for $\mathcal{K}$ on our running example in Figure 2. We start by noting the trivial fact that $\mathcal{OPT}(\mathcal{Q}_{\mathrm{V}})$ must be a subset of $\mathcal{K}$ for (2) to hold. This is indeed the only subset of $\mathcal{Q}_{\mathrm{V}}$ that needs to be *included* in $\mathcal{K}$, and is highlighted in Figure 2 as the blue shaded area.

1. For other costs in (1), we may require other assumptions on $\mathcal{Q}_{\mathrm{V}}$.

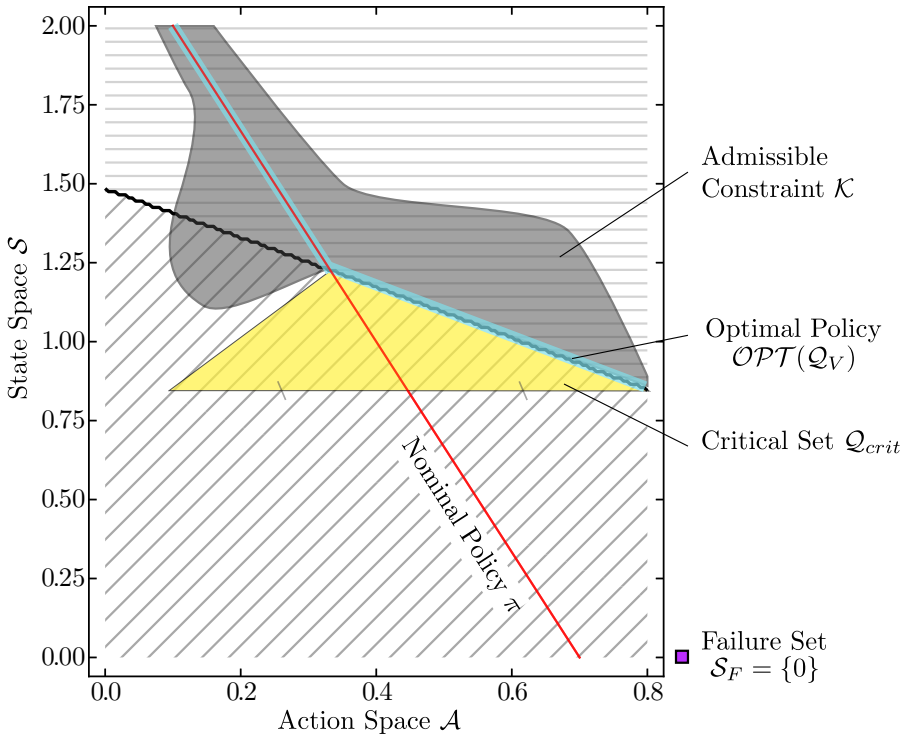

Figure 2: Admissible constraints $\mathcal{K}$ (grey), the set $\mathcal{OPT}(\mathcal{Q}_{\mathrm{V}})$ (blue), and the critical set (yellow) are all defined for a nominal policy π (red line). The optimal policy is the closest to the nominal one in the viable set (horizontal hatch). The critical set is the set of unviable state-actions (diagonal hatch) closer to π than $\mathcal{OPT}(\mathcal{Q}_{\mathrm{V}})$ is. Admissible constraints should include the optimal policy and exclude the critical set. They may contain unviable state-actions that are not critical (grey, diagonal hatch).

It is also necessary to *exclude* the set of state-action pairs that achieve lower cost than $\mathcal{OPT}(\mathcal{Q}_{\mathrm{V}})$ does, but are not in $\mathcal{Q}_{\mathrm{V}}$; we call this the *critical set*:

Definition 5 (Critical Set) *The critical set $\mathcal{Q}_{crit}$ is the set of all state-actions $(s,a) \in \mathcal{Q} \setminus \mathcal{Q}_V$ where $s \in \mathcal{S}_V$ and $J(s,a) \leq J(s, \mathcal{OPT}(\mathcal{Q}_V)(s))$.*

We slightly abuse notations since $J(s,a)$ has the same value for all $a \in \mathcal{OPT}(\mathcal{Q}_{\mathrm{V}})(s)$. This set is highlighted in Figure 2 in yellow. We conclude that it is both sufficient and necessary for a set $\mathcal{K}$ to include $\mathcal{OPT}(\mathcal{Q}_{\mathrm{V}})(s)$ and exclude $\mathcal{Q}_{\mathrm{crit}}$ for (2) to hold:

Theorem 6 *Let $\mathcal{K}$ be a closed set that contains $\mathcal{OPT}(\mathcal{Q}_V)$. Then, $\mathcal{OPT}(\mathcal{K})(s) = \mathcal{OPT}(\mathcal{Q}_V)(s)$ for all $s \in \mathcal{S}_V$ if, and only if, $\mathcal{K} \cap \mathcal{Q}_{crit} = \emptyset$. We call such a $\mathcal{K}$ an* admissible constraint.

The proof for this theorem is in the Appendix A.

3.3. Greed is Good

Suppose we wish to learn an estimate $\hat{\mathcal{K}}$ of an admissible constraint from data with the greedy exploration strategy $\pi_{\hat{\mathcal{K}}}$:

$$\pi_{\hat{\mathcal{K}}}(s) = \begin{cases} \mathcal{OPT}(\hat{\mathcal{K}})(s) & \text{if feasible,} \\ \text{anything} & \text{otherwise.} \end{cases} \tag{3}$$

In the absence of any domain knowledge, a reasonable choice when $\mathcal{OPT}(\hat{\mathcal{K}})$ is infeasible is to simply follow π. In fact, following this exploration strategy naturally ensures that $\mathcal{Q}_{\text{crit}}$ will continue to be explored until the learning agent has excluded it. Unlike in classical reinforcement learning, the exploration requirements are fulfilled by a greedy policy.

4. Results

As a proof of concept, we demonstrate an on-policy exploration strategy combined with a generic constraint learning algorithm with the structure shown in Algorithm 1 to learn an admissible constraint. Specifically, we use the algorithm developed in (Heim et al., 2019), which models the constraint set with a Gaussian process, and is guaranteed to learn the entire viable set $\mathcal{Q}_{\text{V}}$ under the assumptions of an optimistic initialization and infinite sampling of $\mathcal{Q}_{\text{V}}$. However, instead of minimizing variance, the exploration is driven by following the exploration policy $\pi_{\hat{\mathcal{K}}}$. We test two nominal policies: the linear policy used in the running example in all preceding figures, and a stochastic policy that samples from a uniform distribution of actions. These two cases can be viewed as the extremes of an ϵ-greedy exploration strategy, with ϵ set to 0 and 1, respectively. The Python code to reproduce these results is available at: https://github.com/Data-Science-in-Mechanical-Engineering/edge

Algorithm 1 Constraint Set Learner with our Exploration

Input: Initial state s_0, nominal policy π, initial constraint estimate $\hat{\mathcal{K}}$
do
  apply $\pi_{\hat{\mathcal{K}}}(s_i)$: ▷ Greedy On-policy exploration
  if feasible
    $a_i \leftarrow \mathcal{OPT}(\hat{\mathcal{K}})(s_i)$
  else
    $a_i \leftarrow$ anything
  $s_{i+1} \leftarrow T(s_i, a_i)$ ▷ Sample system dynamics
  update the constraint set $\hat{\mathcal{K}}$ ▷ Algorithm specific
  if failed
    re-initialize state
while terminal condition ▷ Algorithm specific

4.1. Linear Nominal Controller

First, we will use the same nominal affine policy in the examples in the preceding sections, with $a = 0.7 - 0.3s$. We choose a rather conservative initialization of $\hat{\mathcal{K}}$ to represent a typical situation in which the nominal policy is designed to be reliable around a given operating point.

We run our algorithm for two batches of ten episodes each, and at the end of each batch we update the hyper-parameters of the constraint learning algorithm. At the beginning of each episode, the agent is initialized in a random state that is in the state space projection of the current $\hat{\mathcal{K}}$. If the agent ever enters the failure set, the episode is immediately terminated, and the last sample is assigned a value of 0. We also terminate episodes after a maximum of 10 steps.

In the end, the agent has collected 196 samples and has learned a safe control policy $\pi_{\hat{\mathcal{K}}}$ that ensures safety for nearly the entire viability kernel. The initial and final estimates of $\hat{\mathcal{K}}$ are shown in Figure 3.

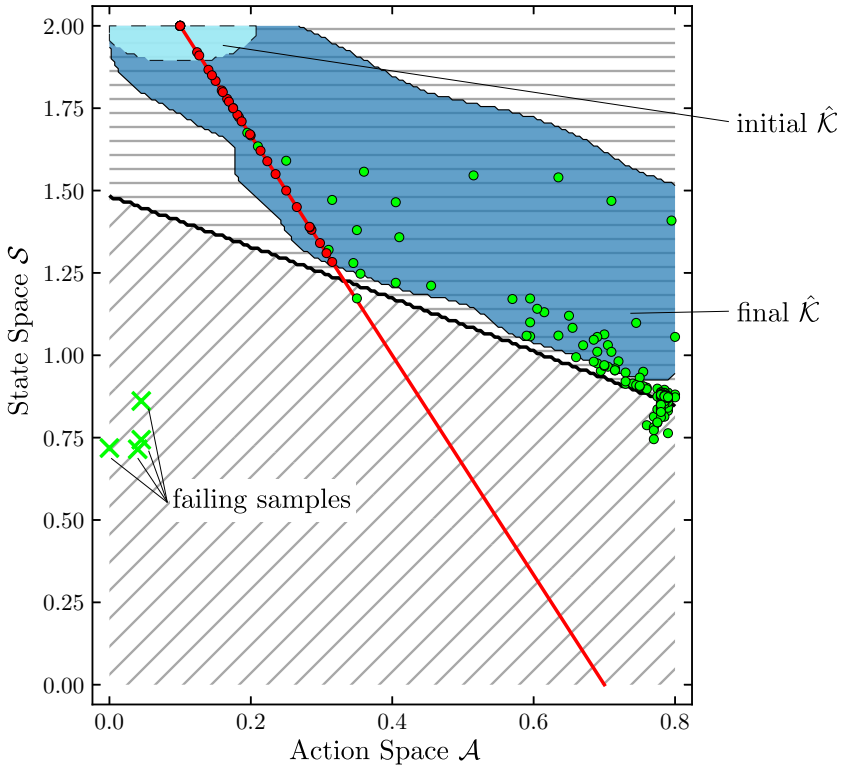

Figure 3: Results for an affine nominal controller (red line) after 20 episodes. The constraint $\hat{\mathcal{K}}$ grows from a cautious initialization (light blue) to an admissible constraint (dark blue). The red dots are steps at which the constraint allowed the nominal controller, and green dots and crosses are the others. The accumulation of samples near the optimal policy show that $\pi_{\hat{\mathcal{K}}}$ converges to it. As expected, the constraint $\hat{\mathcal{K}}$ does not recover the whole viable set.

During the training period, the agent visited the failure set only four times, the last one occurring in episode 15. The learned policy $\pi_{\hat{\mathcal{K}}}$ differs from the lowest-cost policy by at most[2] 10%, and on average 2%, even though $\hat{\mathcal{K}}$ underestimates $\mathcal{Q}_{\mathrm{V}}$ by 43%. This example highlights the advantage of our formulation, as we only explore and learn accurate constraints where necessary.

2. Expressed as percentages of the amplitude of the action set

4.2. Uniformly Random Nominal Controller

Next, we set the nominal policy to sample actions from a uniformly random distribution of all actions, while again running our learning algorithm for two batches of ten episodes and with the same termination criteria. In this case, the critical set contains all state-actions that transition outside of the viability kernel, and the policy continues to push the constraint estimate outwards. Unsurprisingly, $\hat{\mathcal{K}}$ converges towards the true viable set, as can be seen in Figure 4. Here, the discrepancy

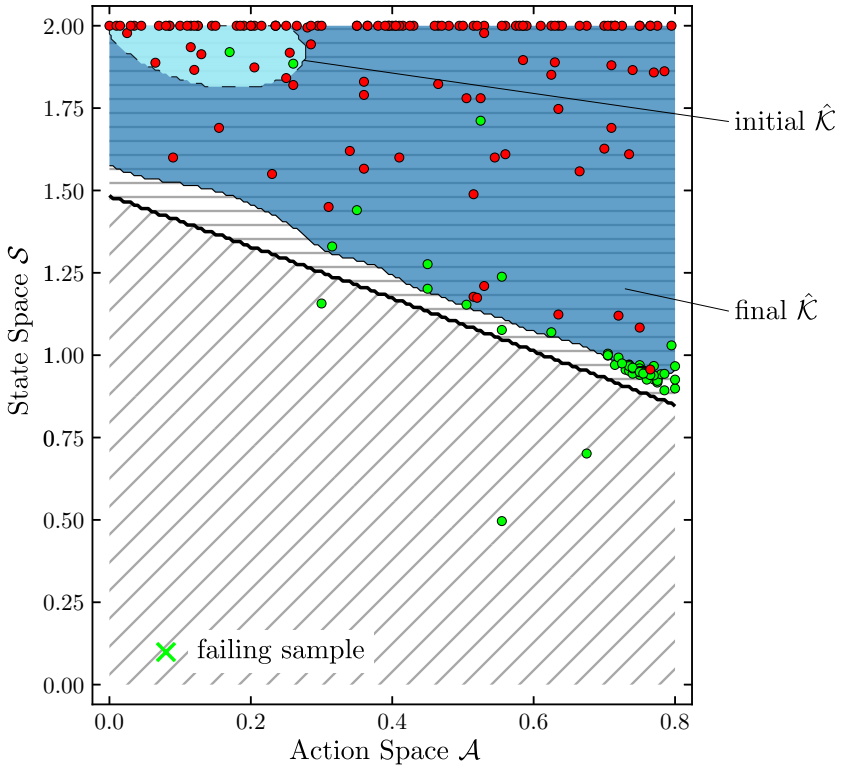

Figure 4: Results for a random nominal controller after 20 episodes. The legend is the same as in Figure 3. Here as well, the constraint $\hat{\mathcal{K}}$ grows to an admissible constraint. Here, the whole viable set is important for the nominal controller, so $\hat{\mathcal{K}}$ recovers it.

between $\pi_{\hat{\mathcal{K}}}$ and the lowest-cost policy is equivalent to how much $\hat{\mathcal{K}}$ underestimates $\mathcal{Q}_{\mathrm{V}}$, since the nominal policy would sample from all of $\mathcal{Q}_{\mathrm{V}}$. In our experiment, this reaches 9% after 197 samples. This example highlights the applicability of our exploration strategy to stochastic policies.

5. Discussion

We have presented a formulation of constrained controllers as sets in state-action space, which allows classes of constraints to be easily defined and analyzed in a static sense, as positive invariance reduces to considering sets and their projection into state space. With this formulation, we identify the *viable set* as the set of all safe state actions, and *control constraints* as all possible subsets that induce positive invariance, and thus guarantee safety over their projection into state space. Finally,

we identify *admissible constraints* as sets used to safely constrain a nominal policy, and show that these are a separate family from control constraints: admissible constraints need not be control constraints. Our key insight is that admissible constraints only need to accurately exclude a *critical set* of state-actions, and that a policy greedily following the nominal policy naturally prioritizes exploring this set. This property also suggests that, when estimating safe sets, it is not necessary to restrict algorithms to conservative estimates.

Our long-term goal is to learn safe control policies, and not just safety constraints for stationary policies. However, how changing the nominal policy affects the minimum sampling requirements needs to be clarified. Although the arguments we have made hold for each instance of a moving policy, the critical set will move together with the policy. We expect that the insights developed in this paper can nonetheless be used to guide exploration and improve sample efficiency. Our focus in this paper has been on a purely data-driven scenario. As is often the case, we also expect that leveraging model-based predictions can further improve sample efficiency and will be critical to scaling up to real-world applications. A straightforward approach is to plan over trajectories (Buisson-Fenet et al., 2020; Hewing et al., 2020). A different approach we find promising is to learn parsimonious models based on the required properties of a control constraint that we have elucidated in Section 2: we do not need models that make accurate state predictions, but only predict if sets map into their projection.

Appendix A. Proof of Theorem 6

We first introduce the *action slice* of $\mathcal{K}$ in state s:

$$\mathcal{K}[s] = \{a \in \mathcal{A}, (s,a) \in \mathcal{K}\}\,.$$

Proof First, suppose that for all $s \in \mathcal{S}_\mathrm{V}$, $\mathcal{OPT}(\mathcal{K})(s) = \mathcal{OPT}(\mathcal{Q}_\mathrm{V})(s)$. We prove that any element (s,a) of $\mathcal{Q}_\mathrm{crit}$ is not in $\mathcal{K}$. Since (s,a) is not in $\mathcal{Q}_\mathrm{V}$, we have:

$$a \notin \operatorname*{argmin}_{b \in \mathcal{K}[s]} J(s,b).$$

Yet, by definition of $\mathcal{Q}_\mathrm{crit}$ as the set of unviable state-actions that achieve lower J than $\mathcal{OPT}(\mathcal{Q}_\mathrm{V})$, we also have:

$$J(s,a) \leq \min_{b \in \mathcal{K}[s]} J(s,b).$$

Combining the last two equations immediately shows that $a \notin \mathcal{K}[s]$, which proves the implication.

Conversely, we prove that $\mathcal{OPT}(\mathcal{K})(s) = \mathcal{OPT}(\mathcal{Q}_\mathrm{V})(s)$ for $s \in \mathcal{S}_\mathrm{V}$. We proceed by proving the double inclusion. Let a in $\mathcal{OPT}(\mathcal{K})(s)$. We immediately have:

$$J(s,a) \leq J(s, \mathcal{OPT}(\mathcal{Q}_\mathrm{V})(s)), \tag{4}$$

since $\mathcal{OPT}(\mathcal{Q}_\mathrm{V})$ is contained in $\mathcal{K}$. Thus, a is at least as close to $\pi(s)$ than $\mathcal{OPT}(\mathcal{Q}_\mathrm{V})(s)$. Therefore, (s,a) is either in $\mathcal{Q}_\mathrm{crit}$ or in $\mathcal{Q}_\mathrm{V}$. Since $\mathcal{K} \cap \mathcal{Q}_\mathrm{crit} = \emptyset$, then (s,a) is in $\mathcal{Q}_\mathrm{V}$, and (4) is an equality. This means that $a \in \mathcal{OPT}(\mathcal{Q}_\mathrm{V})(s)$, and shows the first inclusion.

Now consider a in $\mathcal{OPT}(\mathcal{Q}_\mathrm{V})(s)$. We immediately have $(s,a) \in \mathcal{K}$, since it contains $\mathcal{OPT}(\mathcal{Q}_\mathrm{V})$. Since a is a closest viable actions to $\pi(s)$ and $\mathcal{Q}_\mathrm{V}$ contains $\mathcal{OPT}(\mathcal{K})(s)$, we also have $J(s,a) = J(s, \mathcal{OPT}(\mathcal{K})(s))$. This shows $\mathcal{OPT}(\mathcal{Q}_\mathrm{V})(s) \subset \mathcal{OPT}(\mathcal{K})(s)$, and concludes the proof. ■

Acknowledgments

We thank Friedrich Solowjow and Alexander von Rohr for frequent insightful discussions. This work was funded, in part, by the Cyber Valley Initiative.

References

Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *IEEE European Control Conference (ECC)*, 2019.

Jean-Pierre Aubin, Alexandre M Bayen, and Patrick Saint-Pierre. *Viability theory: new directions*. Springer Science & Business Media, 2011.

S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances. In *IEEE Conference on Decision and Control (CDC)*, 2017.

Nicholas M. Boffi, Stephen Tu, Nikolai Matni, Jean-Jacques E. Slotine, and Vikas Sindhwani. Learning stability certificates from data. In *Conference on Robot Learning (CoRL)*. PMLR, 2020.

Mona Buisson-Fenet, Friedrich Solowjow, and Sebastian Trimpe. Actively learning gaussian process dynamics. In *Learning for Dynamics and Control (L4DC)*. PMLR, 2020.

Konstantinos Chatzilygeroudis, Vassilis Vassiliades, Freek Stulp, Sylvain Calinon, and Jean-Baptiste Mouret. A survey on policy search algorithms for learning robot controllers in a handful of trials. *IEEE Transactions on Robotics*, 36, 2020.

Jaime F. Fisac, Anayo K. Akametalu, Melanie N. Zeilinger, Shahab Kaynama, Jeremy Gillula, and Claire J. Tomlin. A general safety framework for learning-based control in uncertain robotic systems. *IEEE Transactions on Automatic Control*, 64, 2018.

Jaime F. Fisac, Neil F. Lugovoy, Vincenç Rubies-Royo, Shromona Ghosh, and Claire J. Tomlin. Bridging hamilton-jacobi safety analysis and reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, 2019.

David Fridovich-Keil, Sylvia L Herbert, Jaime F. Fisac, Sampada Deglurkar, and Claire J. Tomlin. Planning, fast and slow: A framework for adaptive real-time safe trajectory planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

David Fridovich-Keil, Jaime F. Fisac, and Claire J. Tomlin. Safely probabilistically complete real-time planning and exploration in unknown environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16, 2015.

Steve Heim, Alexander Rohr, Sebastian Trimpe, and Alexander Badri-Spröwitz. A learnable safety measure. In *Conference on Robot Learning (CoRL)*. PMLR, 2019.

Lukas Hewing, Kim P. Wabersich, Marcel Menner, and Melanie N. Zeilinger. Learning-based model predictive control: Toward safe learning in control. *Annual Review of Control, Robotics, and Autonomous Systems*, 3, 2020.

Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 2021.

Shahab Kaynama, John Maidens, Meeko Oishi, Ian M Mitchell, and Guy A Dumont. Computing the viability kernel using maximal reachable sets. In *ACM Conference on Hybrid Systems: Computation and Control*, 2012.

Majid Khadiv, Alexander Herzog, S Ali A Moosavian, and Ludovic Righetti. Walking control based on step timing adaptation. *IEEE Transactions on Robotics*, 36, 2020.

A. Liniger and J. Lygeros. Real-time control for autonomous racing based on viability theory. *IEEE Transactions on Control Systems Technology*, 27, 2019.

Julian Nubert, Johannes Köhler, Vincent Berenz, Frank Allgöwer, and Sebastian Trimpe. Safe and fast tracking on a robot manipulator: Robust mpc and neural network control. *IEEE Robotics and Automation Letters*, 5, 2020.

Saša V. Raković, Fernando A.C.C. Fontes, and Ilya V. Kolmanovsky. Reachability and invariance for linear sampled–data systems. *IFAC World Congress*, 50, 2017.

Benjamin Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2, 2019.

Alexander Robey, Haimin Hu, Lars Lindemann, Hanwen Zhang, Dimos V. Dimarogonas, Stephen Tu, and Nikolai Matni. Learning control barrier functions from expert demonstrations. In *IEEE Conference on Decision and Control (CDC)*, 2020.

Jennifer Shih, Franziska Meier, and Akshara Rai. A framework for online updates to safe sets for uncertain dynamics. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

Pierre-Brice Wieber. Viability and predictive control for safe locomotion. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2018.

Zhaoming Xie, Patrick Clary, Jeremy Dao, Pedro Morais, Jonanthan Hurst, and Michiel Panne. Learning locomotion skills for cassie: Iterative design and sim-to-real. In *Conference on Robot Learning (CoRL)*, 2020.

Zhehua Zhou, Ozgur S. Oguz, Marion Leibold, and Martin Buss. A general framework to increase safety of learning algorithms for dynamical systems based on region of attraction estimation. *IEEE Transactions on Robotics*, 36, 2020.