Proceedings of Machine Learning Research vol 144:1–12, 2021 3rd Annual Conference on Learning for Dynamics and Control

Safe Reinforcement Learning of Control-Affine Systems with Vertex Networks

Liyuan Zheng LIYUANZ8@UW.EDU
University of Washington, Seattle, WA

Yuanyuan Shi YYSHI@ENG.UCSD.EDU
University of California San Diego, San Diego, CA

Lillian J. Ratliff RATLIFFL@UW.EDU
University of Washington, Seattle, WA

Baosen Zhang ZHANGBAO@UW.EDU
University of Washington, Seattle, WA

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

Abstract

This paper focuses on finding reinforcement learning policies for control systems with hard state and action constraints. Despite its success in many domains, reinforcement learning is challenging to apply to problems with hard constraints, especially if both the state variables and actions are constrained. Previous works seeking to ensure constraint satisfaction, or safety, have focused on adding a projection step to the policy during learning. Yet, this approach requires solving an optimization problem at every policy execution step, which can lead to significant computational costs and has no safety guarantee with the projection step removed after training. To tackle this problem, this paper proposes a new approach, termed Vertex Networks (VNs), with guarantees on next step safety during both the exploration and execution stage, by incorporating the safety constraints into the policy network architecture. Leveraging the geometric property that all points within a convex set can be represented as the convex combination of its vertices, the proposed algorithm first learns the convex combination weights and then uses these weights along with the pre-calculated vertices to output an action. Numerical examples illustrate that the proposed VN algorithm outperforms projection-based reinforcement learning methods.

Keywords: safe reinforcement learning, network architecture, convex polytope

1. Introduction

Over the last couple of years, reinforcement learning (RL) algorithms have yielded impressive results on a variety of applications. These successes include playing video games with super-human performance (Mnih et al., 2015), robot locomotion and manipulation (Lillicrap et al., 2015; Levine et al., 2016), autonomous vehicles (Sallab et al., 2017), and many benchmark continuous control tasks (Duan et al., 2016). In RL, an agent learns to make sequential decisions by interacting with the environment, gradually improving its performance at the task as learning progresses.

Policy optimization algorithms (Lillicrap et al., 2015; Schulman et al., 2015) for RL assume that agents are free to explore any behavior during learning, as long as it leads to performance improvement. However, in many real-world applications, there are often additional safety constraints, or specifications that lead to constraints, on the actions or the states of the system. For instance, a

© 2021 L. Zheng, Y. Shi, L.J. Ratliff & B. Zhang.

robot arm should avoid behaviors that could cause it to damage itself or the objects around it, and autonomous vehicles must avoid crashing into others while navigating (Amodei et al., 2016). In fact, constraints are integral to many real-world problems, and maintaining constraint satisfaction during learning is critical. In addition, these are often hard constarints that must be satisfied at all times. Therefore, in this work, our goal is to maintain constraint satisfaction at each step throughout the whole learning process. This problem is sometimes called the *safe RL* problem (Garcıa and Fernández, 2015). In particular, we define safety as staying within some pre-specified safety sets for both states and actions.

In existing safe RL literature, a "safety layer" is often leveraged to maintain constraint satisfaction during training under different problem settings (Wabersich and Zeilinger, 2018a; Dalal et al., 2018; Li et al., 2018). In these approaches, the action of the policy at each time step is not directly executed, but sent to the safety layer which projects it into the required safety region. However, these approaches either involve solving a computationally expensive optimization problem at each time step (Wabersich and Zeilinger, 2018a; Li et al., 2018), or has strict assumptions about the constraints (Dalal et al., 2018). As a matter of fact, if real-time optimization is allowed by the application, then it is often more advantageous to solve a model predictive control (MPC) problem than to ask for a policy learned by RL. Moreover, if the projection is not differentiable, these methods decouple the action with the policy, which may result in an unstable policy with the projection step removed after training.

To alleviate the above limitation of projection-based approaches, we propose Vertex Networks (VNs), where we encode the safety constraints into the policy via neural network architecture design. In VNs, we design a novel safety layer, where instead of solving a projection optimization problem, we compute the vertices of the safety region at each time step and design the action to be the convex combination of those vertices, allowing policy optimization algorithms to explore only the interior of the safe region during training. Therefore we are able to integrate the constraints into the learning process by design and leverage standard policy optimization training techniques.

The contributions of this work can be briefly summarized as follows: (1) To the best of our knowledge, this is the first attempt to encode safety constraints into policies by explicit network design. (2) In simulation on benchmark inverted pendulum and hovercraft control examples, the proposed method achieves better performance as well as constraint satisfaction compared with projection-based methods.

1.1. Related work

Safe RL or learning-based control with safety certification methods have gained significant attention in recent years. A common technique is to employ a "safety layer" or "safety framework" (Akametalu et al., 2014; Fisac et al., 2018; Wabersich and Zeilinger, 2018a,b; Dalal et al., 2018; Li et al., 2018; Cheng et al., 2019; Taylor et al., 2020), which consists of a projection-based supervisory element between the agent and the system. If the action generated by the agent would cause the system to leave the safe region, the safety layer intervenes and projects to the closest action where the system would stay in the safety region. The techniques proposed in (Akametalu et al., 2014; Fisac et al., 2018) are based on a differential game formulation that results in solving a min-max optimal control problem, which can provide the largest possible safe set, but offers very limited scalability. MPC-like schemes are used in (Wabersich and Zeilinger, 2018a,b), which are computationally expensive if executed online. A closed-form solution can be obtained by locally linearizing the learned model (Dalal et al., 2018). However, this approach assumes that only one of the half-space constraints to be violated, which is unlikely to always hold during exploration. These projection-based methods have also been leveraged in the settings where the constraints are defined by control barrier functions (Cheng et al., 2019; Taylor et al., 2020). Special policies can be designed for tailored applications, e.g., power system frequency regulation (Cui and Zhang, 2020, 2021). Finally, most similar to our setting, Li et al. (2018) consider polytope constraints and propose projected DDPG algorithm to maintain safety during learning.

2. Model Setup

Consider a discrete time affine control system in which the system evolves according to

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t) + H(\mathbf{x}_t)\mathbf{u}_t + \mathbf{w}_t, \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{u} \in \mathbb{R}^m$, and f and H are known functions of appropriate dimensions. The unknown disturbance $\mathbf{w}_t$ is assumed to be in a compact set $\mathcal{W}$, where $\mathcal{W} = \{\mathbf{w} | \underline{\mathbf{w}} \leq \mathbf{w} \leq \overline{\mathbf{w}}\}$. Our goal is to maximize the cumulative reward over time horizon T, subject to safety constraints on $\mathbf{x}$ and actuator constraints on $\mathbf{u}$:

$$\max_{\mathbf{u}} \sum_{t=1}^{T} R(\mathbf{x}_t, \mathbf{u}_t) \tag{2a}$$

$$\text{s.t. } \mathbf{x}_{t+1} = f(\mathbf{x}_t) + H(\mathbf{x}_t)\mathbf{u}_t + \mathbf{w}_t \tag{2b}$$

$$\mathbf{x}_t \in \mathcal{X} \tag{2c}$$

$$\mathbf{u}_t \in \mathcal{U}, \tag{2d}$$

where $\mathcal{X}$ and $\mathcal{U}$ are *convex polytopes*. A convex polytope can be defined as an intersection of linear inequalities (half-space representation) or equivalently as a convex combination of a finite number of points (convex-hull representation) (Grünbaum, 2013). This type of constraints are widely used in theory and practice—for example, see (Blanchini and Miani, 2008) and the references within. In this paper, we assume that the problem defined in (2) is feasible for all possible disturbances in $\mathcal{W}$.

The goal of safe RL is to find an optimal feedback controller $\mathbf{u}_t = \pi_\theta(\mathbf{x}_t)$, that maximizes the overall system reward (2a) while satisfies the safety constraints (2c) and the actuator constraints (2d). Solving (2) is a difficult task, even for linear systems with only the actuator constraints, except for a class of systems where analytic solutions can be found (Gokcek et al., 2001). Therefore, RL (and its different variants) have been proposed to search for a feedback controller.

Numerous learning approaches have been adopted to solve the problem when the constraints (2c) and (2d) are not present. However, there are considerably less successful applications of RL to problems with hard constraints. In contrast to the popular projection-based approaches (see Section 1.1), we propose a novel vertex policy network that encodes the *geometry* of the constraints into the network architecture, which makes a differentiable policy that can be trained by any policy optimization algorithm in an end-to-end way. We will discuss the proposed vertex policy network framework in detail in the next section.

3. Vertex Policy Network

The key idea of our proposed Vertex Network (VN) is based on the geometry property of a convex polytope. Given a bounded convex polytope $\mathcal{P}$, it is always possible to find a finite number of *vertices* such that the convex hull is $\mathcal{P}$. In addition, there is no smaller set of points whose convex hull forms $\mathcal{P}$ (Grünbaum, 2013). Then, the next proposition follows directly.

Proposition 1 *Let $\mathcal{P}$ be a convex polytope with vertices $P_1, \ldots, P_N$. For every point $\mathbf{p} \in \mathcal{P}$, there exists $\lambda_1, \ldots, \lambda_N$, such that*

$$\mathbf{p} = \lambda_1 P_1 + \cdots + \lambda_N P_N,$$

where $\lambda_i \geq 0, \forall i$ and $\lambda_1 + \cdots + \lambda_N = 1$.

The preceding proposition implies that we can search for the set of weights λ_i's instead of directly finding a point inside polytope.

Proposition 1 can be applied to find a feedback control policy. Since both the constraint sets $\mathcal{X}, \mathcal{U}$ and $\mathcal{W}$ are convex polytopes, the feasible control action at each time step must also live in a convex polytope. If its vertices are known, it suffices for the policy to learn the weights λ_i's. The benefit of having the weights as the output is threefold. Firstly, it is much easier to normalize a set of real numbers to be all positive and to sum to unity (the probability simplex) than to project into an arbitrary polytope. For this paper, we use a softmax layer for this purpose. Secondly, this approach allows us to fully explore the interior of the feasible space, where projections could be biased towards the boundary of the set. Thirdly, we are able to use standard policy optimization training techniques by plugging in this policy network design.

For general policy optimization algorithm, the parameterized policy neural network π_θ is updated by

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta), \tag{3}$$

where $J(\theta)$ is the expected return of the current policy and is defined by

$$J(\theta) = \mathbb{E}\left[\sum_{t=1}^{T} R(\mathbf{x}_t, \mathbf{u}_t)\right]. \tag{4}$$

$J(\theta)$ is often approximated by $\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} R(\mathbf{x}_{i,t}, \mathbf{u}_{i,t})$, where N are the number of sampled trajectories generated by running the current policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ and T is the trajectory length. The overall algorithm procedure with the proposed VN framework is provided in Fig. 1.

Below, we discuss the two major components of VN in detail: (1) the safety region and vertex calculation, and (2) the neural network architecture design for the safety layer.

3.1. Evolution of the Action Constraint Set

We require that at each time step the states of the system stay in the set $\mathcal{X}$ subject to all possible disturbances in $\mathcal{W}$, and the control actions at constrained to be in the set $\mathcal{U}$. As stated earlier, we assume $\mathcal{X}, \mathcal{U}$ and $\mathcal{W}$ to be convex polytopes. The main algorithmic challenge comes from the need to repeatedly intersect translated versions of these polytopes. To be concrete, suppose we are given $\mathbf{x}_t$. Then for the next step, we require that $\mathbf{x}_{t+1} \in \mathcal{X}$. This translates into an affine constraint on $\mathbf{u}_t$, since the control action mush satisfy

$$H(\mathbf{x}_t)\mathbf{u}_t \in \mathcal{X} \ominus \mathcal{W} - f(\mathbf{x}_t). \tag{5}$$

where $\ominus$ represents Pontryagin set difference, and $\mathcal{X} \ominus \mathcal{W} = \{\mathbf{x}|\mathbf{x} + \mathcal{W} \subseteq \mathcal{X}\}$ is a polytope. Since $\mathbf{x}_t$ is known, $H(\mathbf{x}_t)$ is a constant matrix in the above equation, and the constraint on $\mathbf{u}_t$ imposed by (5) is again polytopic. We denote this polytope as $\mathcal{S}_t$. The set to which $\mathbf{u}_t$ must belong is the intersection of $\mathcal{S}_t$ and the actuator constraints:

$$\mathcal{U}_t = \mathcal{S}_t \cap \mathcal{U}. \tag{6}$$

After identifying the vertices of $\mathcal{U}_t$, the algorithm in Fig. 1 can be used to find the optimal feedback policies.

Remark 2 *Note that the safety requirement here is "one step ahead". One could use the methods in (Gilbert and Tan, 1991; Kolmanovsky and Gilbert, 1998) to compute a tighter "infinity steps ahead" invariant set $\mathcal{S}_t$ for linear systems. See (Blanchini, 1999) for more discussions on controlled invariant set. However, this is computationally more complicated and out of the scope of this work.*

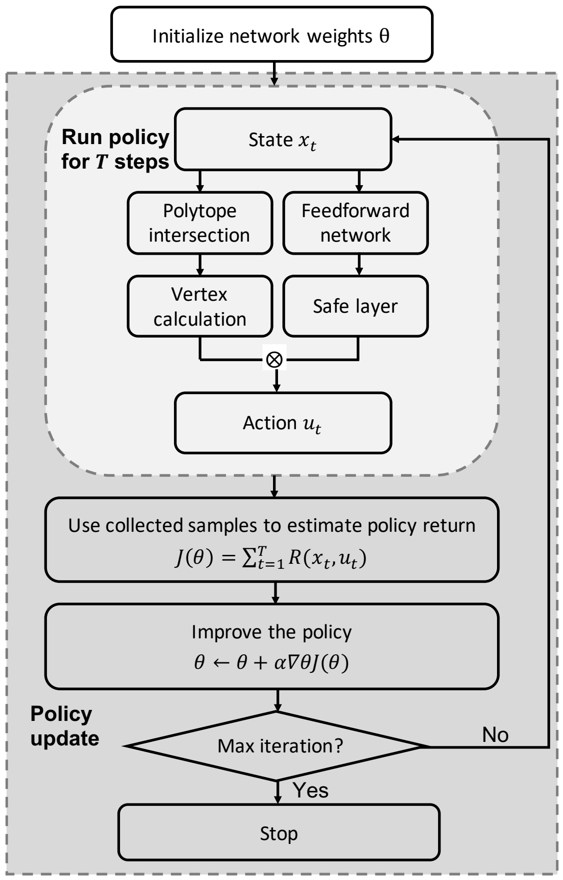

Initialize network weights θ
Run policy for T steps
State x_t
Polytope intersection
Feedforward network
Vertex calculation
Safe layer
⊗
Action u_t
Use collected samples to estimate policy return
$J(\theta) = \sum_{t=1}^{T} R(x_t, u_t)$
Improve the policy
$\theta \leftarrow \theta + \alpha \nabla\theta J(\theta)$
Policy update
Max iteration?
No
Yes
Stop

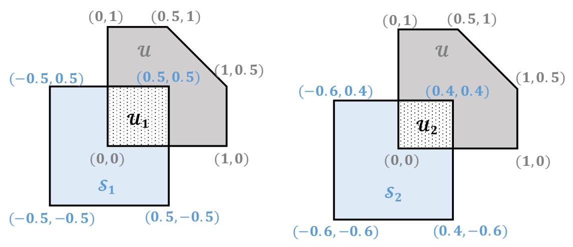
Figure 1: Flowchart of policy optimization with the proposed VN framework.

In general, it is fairly straightforward to find the convex hull or the half-space representations of $\mathcal{S}_t$, since it just requires a linear transformation of $\mathcal{X} \ominus \mathcal{W}$. However, the intersection step in (6) and extracting the vertices are non-trivial. Below, we walk through a simple example to illustrate the steps and then discuss how to overcome the computational challenges in Section 3.2.

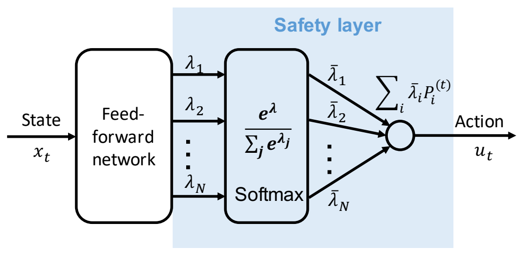
Example 1 (Intersection Step) *Consider the following two–dimensional linear system:*

$$\mathbf{x}_{t+1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{x}_t + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{u}_t.$$

Suppose the action safety set $\mathcal{U}$ is a convex polytope defined by: $0 \leq u_1 \leq 1$, $0 \leq u_2 \leq 1$ and $u_1 + u_2 \leq 1.5$. The state safety set $\mathcal{X}$ is a square defined by $0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1$ and the initial state is $\mathbf{x}_1 = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}^T$. By simple calculation, $\mathcal{S}_1 = \{-0.5 \leq u_1 \leq 0.5, -0.5 \leq u_2 \leq 0.5\}$ and $\mathcal{U}_1$ is the box bounded by $\{(0,0),(0,0.5),(0.5,0),(0.5,0.5)\}$. Fig. 2(a) (left) visualizes the intersection operations. Now suppose a feasible action $\mathbf{u}_1 = [0.1\ 0.1]^\top$ is chosen and the system evolves. Then, $\mathcal{S}_2 = \{-0.6 \leq u_1 \leq 0.4, -0.6 \leq u_2 \leq 0.4\}$. Performing the intersection of $\mathcal{S}_2$ and $\mathcal{U}$, we get that $\mathcal{U}_2$ is a rectangle defined by the vertices $\{(0,0),(0,0.4),(0.4,0),(0.4,0.4)\}$ as depicted in Fig. 2(a) (right).



(a)



(b)

Figure 2: (a) Evolution of action constraint set of Example 1. The left plot visualizes the safety set at time $t = 1$, and the right plot shows the safety set at time $t = 2$. (b) Illustration of the proposed safety layer architecture. The output of the policy network is modified to predict the weights $\lambda_i, i \in [1, N]$. These weights are normalized to $\bar{\lambda}_i, i \in [1, N]$ that satisfies $\bar{\lambda}_i \geq 0$ and $\sum_{i=1}^N \bar{\lambda}_i = 1$, via the softmax activation function. The action output is calculated as $\sum_{i=1}^N \bar{\lambda}_i P_i^{(t)}$, where $P_i^{(t)}$ are the safety polytope vertices.

3.2. Intersection of Polytopes

It should be noted that finding the vertices of an intersection of polytopes is not easy (Tiwary, 2008). If the polytopes are in half-space representation, then their intersection can be found by simply stacking the inequalities. However, finding the vertices of the resulting polytope can be computationally expensive. Similarly, directly intersecting two polytopes based on their convex-hull representation is also intractable in general.

Luckily, in many applications, we are not intersecting two generic polytopes at each step. Rather, as illustrated in Example 1, they are two polytopes with fixed shapes, where the position of $\mathcal{U}$ is fixed as well and only the position of $\mathcal{S}_t$ depends on $\mathbf{x}_t$. It turns out that for many systems (see Section 4), we can find the resulting vertices by hand-designed rules. In addition, there are heuristics that work well for low-dimensional systems (Broman and Shensa, 1990). Applying the proposed VN technique to high-dimensional systems is the main future direction for this work.

For the case that $\mathcal{S}_t$ and $\mathcal{U}$ do not overlap, we pick the point in $\mathcal{U}$ that closest to $\mathcal{S}_t$ to be the vertex. By design, the output of the VN is the action within set $\mathcal{U}$, meanwhile transiting to the state closest to the safe state set $\mathcal{X}$. One could also choose to terminate the current training episode and reset the environment.

3.3. Safety Layer

Once we obtain $\mathcal{U}_t$, the next step is to encode the geometry information into the policy network such that the generated action stays in $\mathcal{U}_t$. According to Proposition 1, it suffices for the policy network to generate the weights (or coefficients) of that convex combination.

Suppose that $\mathcal{U}_t$ can have at most N vertices, labeled $P_1^{(t)}, \ldots, P_N^{(t)}$. In the policy network architecture design, we add an intermediate safety layer that first generates N nodes $\lambda_1, \ldots, \lambda_N$. The value of these nodes, however, are not positive nor do they sum to 1. Therefore, a softmax unit is included as the activation function in order to guarantee the non-negativity and the summation constraints. In particular, we define $\bar{\lambda}_i = e^{\lambda_i}/(\sum_{j=1}^N e^{\lambda_j})$, the weights of a convex combination.

The final output layer (action u_t) is defined as the multiplication of these normalized weights $\bar{\lambda}_i$ and the corresponding vertex values,

$$\mathbf{u}_t = \sum_{i=1}^{N} \bar{\lambda}_i P_i^{(t)}. \tag{7}$$

An illustration diagram is provided in Fig. 2(*b*).

Note that the exact number of vertices of the polytope intersection may vary from state to state. In our design, we simply set N to be the largest possible number, since the convex combination with repeated vertices still captures all points in the polytope. Also, our method relies on an offline hand-designed rule to compute the vertices. Such hand-designed rule is a mapping from state to a sequence of vertices, which is "standardized" as the vertices are in a fixed order.

4. Simulation

In this section, we present and analyze the performance of the proposed VN experimentally. We first describe the baseline algorithms and then demonstrate the performance comparisons in two benchmark control tasks: (i) inverted pendulum and (ii) hovercraft tracking. The code for our experiments is available at `https://github.com/LeoZhengZLY/vertex-net`.

4.1. General Experimental Details

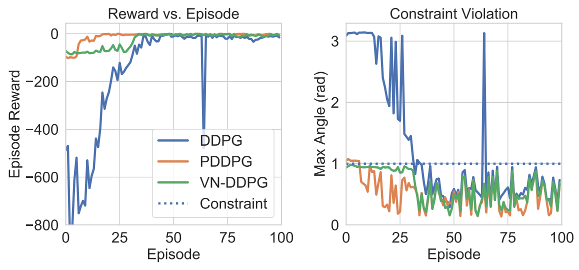
Baseline Learners and Comparators. In our simulations, we compare against two baseline algorithms. (1) We compare against DDPG algorithm (Lillicrap et al., 2015), which is a state-of-the-art continuous control RL algorithm, to optimize the controller policy. To add safety constraints to the vanilla DDPG algorithm, a natural approach is to artificially shape the reward such that the agent will learn to avoid undesired areas. This can be done by setting a low or negative reward to the unsafe states and actions (termed as penalty method in (Garcıa and Fernández, 2015)). In our experiments, we include such a soft penalty in the reward function and train a standard policy network with DDPG algorithm as the first baseline.[1] (2) We also compare against the projected DDPG algorithm in (Li et al., 2018) as the second baseline algorithm (refer to as PDDPG). This approach introduces a projection-based supervisory element between the agent and the system, which projects the unsafe actions into the safety set at each time step during training. According to (Li et al., 2018), after the training, the supervisor is removed and the agent acts as a conventional RL-based controller.

Implementation Details. We use the following hyperparameters for all experiments. For DDPG and PDDPG, we use a three-layer feed-forward neural network, with 256 nodes in each hidden layer. For our approach, we use the proposed VN to represent the policy and leverage DDPG to optimize it directly (refer to as VN-DDPG). It has two feed-forward layers (with 256 nodes in each hidden layer) and a final safety layer as described in Section 3.3.

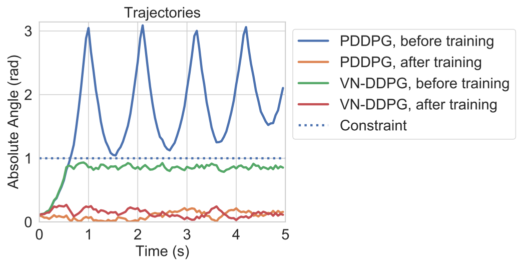
4.2. Pendulum

Experiment Setup. For the inverted pendulum problem, we use the OpenAI gym environment (`pendulum-v0`), with the following specifications: mass $m = 1$, length $l = 1$. The system state is two-dimension that include angle θ and angular velocity ω of the pendulum, and the control

1. The reward function is consistent among all methods, meaning that our method also includes such soft penalty.



(*a*)



(*b*)

Figure 3: (a) Comparison of accumulated reward and constraint violation (max angle) for the pendulum problem. (b) Representative trajectories (angle vs. time) generated by the policy before training and after training.

variable is the applied torque u. We set the safe region to be $\theta \in [-1, 1]$ (radius) and torque limits $u \in \mathcal{U} = [-15, 15]$. The reward function is defined as $r = -(\theta^2 + 0.1\omega^2 + 0.001u^2)$, with the goal of learning an optimal feedback controller. With a discretization step size of $\Delta = 0.05$, the following are the discretized system dynamics:

$$\theta_{t+1} = \theta_t + \omega_t \Delta + \frac{3g}{2l}\sin(\theta_t)\Delta^2 + \frac{3}{ml^2}u\Delta^2 + d_t \tag{8a}$$

$$\omega_{t+1} = \omega_t + \frac{3g}{2l}\sin(\theta_t)\Delta + \frac{3}{ml^2}u\Delta \tag{8b}$$

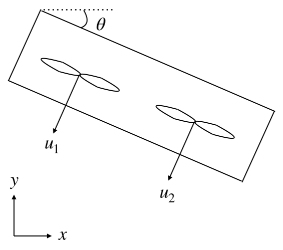
where d_t is the disturbance random variable uniformly drawn from $[-0.05, 0.05]$.

Technical Details. To keep the next state in the safe region $\theta_{t+1} \in [-1, 1]$, we can compute the corresponding upper and lower bound of u to represent set $\mathcal{S}_t$ by (8a). Therefore, the vertices of VN can be found by intersecting $\mathcal{S}_t$ and $\mathcal{U}$. Under the case where $\mathcal{S}_t$ and $\mathcal{U}$ have no overlap, we pick -15 as the vertices if the upper bound of $\mathcal{S}_t$ is less than -15. Otherwise, we pick 15 as the vertices. For comparison, the output of normal policy network is constrained in $[-15, 15]$ by using `tanh` activation function in the final layer. The initial state of each episode is randomly sampled in the safe state region $[-1, 1]$.

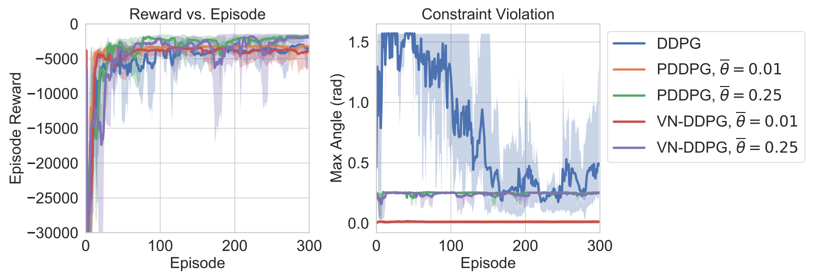
Results. In Fig. 3(*a*), we show a comparison of the accumulated reward and the max angle of each episode in the training of DDPG, PDDPG, and VN-DDPG. We observe that both PDDPG and VN-DDPG maintain safety throughout the training process, and as a result, it achieves higher reward in the early stage. It is also interesting to observe that the DDPG also becomes "safe" after training, since the reward function itself drives θ to be small. This suggests if we can train the DDPG with soft penalty offline, it might be able to obey the safety constraint for some control systems. However, the plot on max angle shows that even a well-trained policy may occasionally violate constraints if these hard constraints are not explicitly taken into account.

Fig. 3(*b*) shows the pendulum angle trajectories of representative episodes before training versus after training generated by PDDPG and VN-DDPG. For VN-DDPG before training, the pendulum angle is maintained near the edge of the safe region.[2] However, the PDDPG fails to maintain safety

2. Since we initialize the neural network near to zero, the policy before training is simply the average of all vertices.



(a)



(b)

Figure 4: (a) Hovercraft example. u_1 and u_2 denote starboard and port fan forces. θ, x, y are the tilt angle and the coordinate position. (b) Comparison of accumulated reward and constraint violation (max tile angle) from Hovercraft control with different constraint upper bound.

with the projection supervisor removed. Once the training is finished, both of the learned controllers are safe. These suggest that PDDPG could be unsafe if the training has not converged but VN-DDPG maintains safety regardless of the neural network initialization.

4.3. Hovercraft

Experiment Setup. Consider the task of controlling a hovercraft that tracks a target position illustrated in Fig. 4(*a*). The system dynamics are defined as follows (Hespanha, 2018):

$$x_{t+1} = x_t + v_{x,t}\Delta + \frac{1}{2m}\sin\theta_t(u_1+u_2)\Delta^2 \tag{9a}$$

$$v_{x,t+1} = v_{x,t} + \frac{1}{m}\sin\theta_t(u_1+u_2)\Delta \tag{9b}$$

$$y_{t+1} = y_t + v_{y,t}\Delta + \frac{1}{2m}(\cos\theta_t(u_1+u_2)-g)\Delta^2 \tag{9c}$$

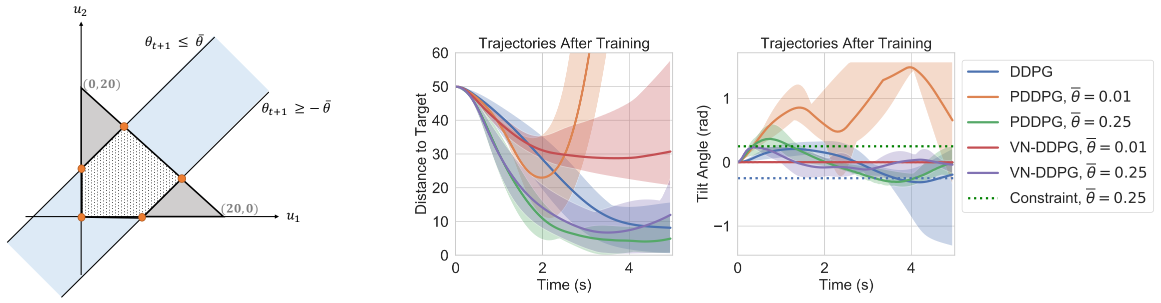
$$v_{y,t+1} = v_{y,t} + \frac{1}{m}(\cos\theta_t(u_1+u_2)-g)\Delta \tag{9d}$$

$$\theta_{t+1} = \theta_t + v_{\theta,t}\Delta + \frac{1}{2l}(u_1-u_2)\Delta^2 \tag{9e}$$

$$v_{\theta,t+1} = v_{\theta,t} + \frac{1}{l}(u_1-u_2)\Delta \tag{9f}$$

where $m = l = 1, g = 10$. The initial state of the hovercraft is set at position $(0,0)$ and the target position is $(5,5)$. To keep the tilt angle of the hovercraft in a safety region, we set the safe state region to be $\theta \in [-\overline{\theta}, \overline{\theta}]$. To better investigate the effect of the constraint, we did two experiments where the tilt angle upper bounds are set to be $\overline{\theta} = 0.01$ and $\overline{\theta} = 0.25$ radians, respectively. Considering the force exerted on two fans are coupled and subject to a total energy budget, the actuator constraint set is defined as $\mathcal{U} = \{u_1, u_2 | u_1 \geq 0, u_2 \geq 0, u_1 + u_2 \leq 20\}$.

Technical Details. For all the learners, we define the reward function $r = -(x-x_0)^2 - (y-y_0)^2 - \theta^2 - 0.1(v_x^2 + v_y^2 + v_\theta^2) - 0.001(u_1^2 + u_2^2)$ where (x_0, y_0) denotes the target position. The learner is incentivized to track the target position while keeping the tile angle and force small for the safety constraint. In addition, for DDPG and PDDPG, the output of policy network is truncated



(a) (b)

Figure 5: (a) Illustration of polytope intersection of hovercraft examples. (b) Trajectories (distance to target $||(x, y) - (x_0, y_0)||^2$ and tilt angles θ) of trained policies with different $\bar{\theta}$.

to be within the actuator constraint set $\mathcal{U}$. For the proposed VN-DDPG approach, Fig. 5(a) shows how to use at most five vertices to represent the intersection of safe state region and the actuator constraint region. The gray area is the actuator constraint set $\mathcal{U}$, the blue area are the constraint on u_1, u_2 imposed by $\theta_{t+1} \in [-\overline{\theta}, \overline{\theta}]$ and (9e). The orange vertices can be computed in closed-form by intersecting boundary equations.

Results. Fig. 4(b) compares the accumulated reward and max tilt angle of each episode in the training of DDPG, PDDPG, and VN-DDPG. We observe that even trained DDPG could still constantly violate the constraint. Both PDDPG and VN-DDPG are able to maintain constraint satisfaction during training and we can observe that when $\overline{\theta} = 0.01$, they converge to suboptimal controllers with lower episode reward since the constraint is too restricted and contradicts with the task (tracking the target position). Fig. 5(b) visualizes the trajectories of trained policies. In both choices of the tilt angle upper limit $\overline{\theta}$, the constraint is never violated in the whole trajectory executing learned policies of VN-DDPG. When $\overline{\theta} = 0.01$, the hovercraft has a strict constraint on its tilt angle and fails to track the target position but with $\overline{\theta} = 0.25$, it is able to reach the target. On the contrary, running the learned DDPG and PDDPG policies will have tilt angle violation even with the soft penalty added in the reward. Note that the trained policy of PDDPG with $\overline{\theta} = 0.01$ has poor performance as it completely fails to track the target and violates the constraint. This is due to the fact that the constraint during training is too restricted and the projected actions deviate a lot from the original unconstrained policy. As a matter of fact, the decoupling of the action and the policy results in an unstable policy with the projection step removed after training.

5. CONCLUSIONS

We design a novel policy network architecture called Vertex Network (VN), which is motivated by the problem of training an RL algorithm with hard state and action constraints. Leveraging the geometric property that a convex polytope can be equivalently represented as the convex hull of a finite set of vertices, the output of VN satisfies the safety constraints by design. Empirically, we show that VN yields significantly better safety performance compared with a normal policy network with a constraint violation penalty or with a projection-based supervisor in two benchmark control systems.

References

Anayo K Akametalu, Jaime F Fisac, Jeremy H Gillula, Shahab Kaynama, Melanie N Zeilinger, and Claire J Tomlin. Reachability-based safe learning with gaussian processes. In *53rd IEEE Conference on Decision and Control*, pages 1424–1431. IEEE, 2014.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

Franco Blanchini. Set invariance in control. *Automatica*, 35(11):1747–1767, 1999.

Franco Blanchini and Stefano Miani. *Set-theoretic methods in control*. Springer, 2008.

V Broman and MJ Shensa. A compact algorithm for the intersection and approximation of n-dimensional polytopes. *Mathematics and computers in simulation*, 32(5-6):469–480, 1990.

Richard Cheng, Gábor Orosz, Richard M Murray, and Joel W Burdick. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3387–3395, 2019.

Wenqi Cui and Baosen Zhang. Reinforcement learning for optimal frequency control: A lyapunov approach. *arXiv preprint arXiv:2009.05654*, 2020.

Wenqi Cui and Baosen Zhang. Lyapunov-regularized reinforcement learning for power system transient stability. *arXiv preprint arXiv:2103.03869*, 2021.

Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

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(7):2737–2752, 2018.

Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

Elmer G Gilbert and K Tin Tan. Linear systems with state and control constraints: The theory and application of maximal output admissible sets. *IEEE Transactions on Automatic control*, 36(9): 1008–1020, 1991.

Cevat Gokcek, Pierre T Kabamba, and Semyon M Meerkov. An lqr/lqg theory for systems with saturating actuators. *IEEE Transactions on Automatic Control*, 46(10):1529–1542, 2001.

Branko Grünbaum. *Convex polytopes*, volume 221. Springer Science & Business Media, 2013.

Joao P Hespanha. *Linear systems theory*. Princeton university press, 2018.

Ilya Kolmanovsky and Elmer G Gilbert. Theory and computation of disturbance invariant sets for discrete-time linear systems. *Mathematical problems in engineering*, 4, 1998.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Zhaojian Li, Uroš Kalabić, and Tianshu Chu. Safe reinforcement learning: Learning with supervision using a constraint-admissible set. In *2018 Annual American Control Conference (ACC)*, pages 6390–6395. IEEE, 2018.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

Andrew Taylor, Andrew Singletary, Yisong Yue, and Aaron Ames. Learning for safety-critical control with control barrier functions. In *Learning for Dynamics and Control*, pages 708–717. PMLR, 2020.

Hans Raj Tiwary. On the hardness of computing intersection, union and minkowski sum of polytopes. *Discrete & Computational Geometry*, 40(3):469–479, 2008.

Kim P Wabersich and Melanie N Zeilinger. Linear model predictive safety certification for learning-based control. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 7130–7135. IEEE, 2018a.

Kim P Wabersich and Melanie N Zeilinger. Safe exploration of nonlinear dynamical systems: A predictive safety filter for reinforcement learning. *arXiv preprint arXiv:1812.05506*, 2018b.