

Appendices

A Frequently Asked Questions

We discuss some design choices and implications of our method using an informal FAQ format.

What does it mean for the proposed gameplay filter to approximate a perfect filter? If we had the exact solution to the Isaacs reach-avoid equation (5), our gameplay rollouts would be necessary and sufficient for safely reaching \mathcal{T} in H (or fewer) steps. Since \mathcal{T} is typically chosen to be a broad, naturally reachable class of robot states (e.g., coming to a stable stance for a walking robot or pulling over for an autonomous vehicle), safely reaching \mathcal{T} within a long enough horizon H is possible as long as remaining safe is possible in the first place. In other words, the sufficient reach-avoid condition becomes a “tight” approximation of the all-time safety condition. We can observe this phenomenon in Fig. 3, where the reach-avoid filter’s overstepping vanishes with long H .

Why is reach-avoid preferable if it’s more conservative than avoid-only? This is an important aspect of predictive safety filtering and relates to a deeper tenet in safety engineering philosophy: whatever the safety boundary is (a strategy that is “just safe enough”) is, it is preferable to approach it from the safe side than from the unsafe side. In practice, we don’t know a priori how many steps we need to avoid being blindsided by future failures just beyond the lookahead horizon. When in doubt it’s preferable to risk being overly conservative than to risk losing safety.

Having a terminal state constraint is common in MPC, how is reach-avoid different? The use of a terminal controlled-invariant set in MPC is well established and ensures recursive feasibility. Our choice of reach-avoid over an avoid-only safety condition is an instance of the same principle. An important difference is that the (also well-established) reach-avoid condition gives our filter extra flexibility by allowing the gameplay trajectory to reach the forever-safe set at any time within the horizon. This reduces conservativeness and often lets us to terminate the gameplay rollout early.

How do you determine \mathcal{T} ? In practice, a suitable \mathcal{T} is obtained from domain knowledge, offline computation, pre-deployment learning, or some combination, often in the form of a stability basin (or region of attraction) around a desirable class of equilibrium points sufficiently away from failure. For example, most robots can be robustly stabilized around static or steady cruising configurations by comparatively simple linear feedback controllers (e.g., most modern walking robots ship with built-in controllers that can stabilize them around a default stance). Larger all-time safe regions may be found by (robust) Lyapunov analysis or even optimized through control Lyapunov functions.

What are the implications of the choice of \mathcal{T} ? Broadly speaking, the larger the \mathcal{T} we can characterize offline, the easier the job of the gameplay filter at runtime, and, potentially, the fewer steps we’ll need to reach it from more dynamic configurations. In fact, in the extreme case, we could be remarkably lucky and find $\mathcal{T} = \Omega^*$, in which case, the gameplay filter’s job is made much easier, since all candidate actions that are safe will keep the state in \mathcal{T} , immediately terminating the rollout check. Conversely, all actions that leave \mathcal{T} are unsafe and the gameplay rollout will not be able to return to \mathcal{T} . In order to avoid initializing the gameplay filter from a no-win scenario, designers should ensure that \mathcal{T} contains the range of expected robot deployment conditions (\mathcal{X}_0) in the ODD.

Why aren’t you using onboard cameras or lidar? Our empirical focus in this paper is on demonstrating automatically synthesized safety filters that account for the full-order (36-D) walking dynamics of quadruped robots. We think that the simplest and clearest demonstration of this concept is by having the filter only consider the robot’s own state (proprioception) without accounting for the environment, obstacles, etc. (exoeption). That said, incorporating information about the robot’s surroundings can be extremely valuable—and often critical—to safety. We are very excited by the scalability and generality that new safety approaches like the one we present in this paper seem to enjoy, and we expect they will soon unlock full-order safety filters that incorporate rich exoeceptive information in real time, whether straight from raw sensor data or through intermediate representations provided by the perception and localization stack.

B Implementation Details

State and action spaces. For the scope of this paper, we aim to construct a *proprioceptive* safety filter that relies on onboard estimation of the robot’s kinematic state but *no exoceptive* information (from camera, lidar, etc.) about the surrounding environment.⁴ We encode the quadrupedal robots’ state and action vectors as follows:

$$\begin{aligned} x &:= [p_x, p_y, p_z, v_x, v_y, v_z, \theta_x, \theta_y, \theta_z, \omega_x, \omega_y, \omega_z, \{\theta_j^i\}, \{\omega_j^i\}] , \\ u &:= [\{\delta\theta_j^i\}] , \end{aligned}$$

with p_x, p_y, p_z the position of the body frame with respect to a fixed reference (“world”) frame; v_x, v_y, v_z the velocity of the robot’s torso expressed in (forward–left–up) body coordinates; $\theta_x, \theta_y, \theta_z$ the roll, pitch, and yaw angles of the robot’s body frame with respect to the world frame;⁵ $\omega_x, \omega_y, \omega_z$ the body frame’s axial rotational rates; and $\theta_j^i, \omega_j^i, \delta\theta_j^i$ the angle, angular velocity, and commanded angular increment of the robot’s i^{th} joint.

The above constitutes a full-order state representation of the robot’s idealized Lagrangian mechanics. A total of 18 generalized coordinates encode the 6 degrees of freedom of the torso’s rigid-body pose in addition to the configuration of 3 rotational joints (hip abduction, hip flexion, and knee flexion) for each of the 4 legs; the robot’s rate of motion is expressed through 18 corresponding generalized velocities, for a total 36 continuous state variables. We discuss discrete contact variables below.

The robot’s control authority is achieved by independently modulating the torque applied on each of its 12 rotational joints by an electric motor; in modern legged platforms, these motors typically have dedicated low-level controllers, so our control policy sends a tracking reference to each motor controller rather than directly commanding a torque.

Finally, the disturbance is modeled as an external force that can act on any point of the robot’s body and in any direction of Euclidean space, with a bounded modulus. The specified range of admissible disturbance forces is discussed below.

Black-box simulator(s). The dynamical model is implemented by the off-the-shelf PyBullet physics engine [52] using the standardized robot description files made available by the manufacturers of each platform. Our method treats the simulator as black-box environment for both training and runtime safety filtering, allowing the engine and/or robot model to be easily swapped out. The generality and modularity of our approach is perhaps best illustrated by the fact that we synthesized and deployed the safety filter for the Go-2 robot using *identical hyperparameter values* as for the S40 robot. Our only modification, other than replacing the robot model in the physics engine, was to append 4 state components to each neural network’s input space to account for foot contact information; we note that even this straightforward addition is entirely optional, since we could have alternatively constructed a safety filter that simply disregarded the extra sensor data.

Actor and critic networks. The learned control and disturbance actors, as well as the safety critics, are independent of the robot’s absolute position p_x, p_y, p_z and heading angle θ_z ; of these, only distance to the ground has an effect on the dynamics, but since it is hard to observe without vision, we do not make it available. In the case of the Go-2 quadruped (but not the S40), the policies additionally depend on the discrete contact state, encoded as a Boolean (true/false) indicator for each foot. In simulation, each neural network policy receives as input the ground-truth state of the robot in the simulator; in hardware experiments, they instead receive a state estimate computed by the robot’s on-board perception stack. Each policy is implemented by a fully-connected feedforward neural network with 3 hidden layers of 256 neurons, and critics have 3 hidden layers with 128 neurons.

⁴Ranged perception can improve the robustness of walking controllers by sensing terrain geometry and texture, and it is strictly needed for ODDs including unmapped or moving obstacles. Full-order legged robot safety filters combining proprioception and exoception have significant potential and are ripe for investigation.

⁵For the purposes of this demonstration, we find that an Euler angle representation of body attitude performs adequately and makes the failure set straightforward to encode. In general, a quaternion-based representation may be preferable, avoiding the risk of computational issues in the neighborhood of singularities (at $\theta_y = \pm \frac{\pi}{2}$).

Table 5: Implementation details of the safety specifications of quadruped walking.

| Notation | Magnitude |
|--------------------------------|-----------|
| $\bar{z}_{\text{corner},g}$ | 0.1 m |
| \bar{z}_{knee} | 0.05 m |
| $\bar{z}_{\text{corner},\ell}$ | 0.4 m |
| \bar{z}_{foot} | 0.05 m |
| $\bar{\omega}$ | 10 deg/s |
| \bar{v} | 0.2 m/s |

Safety specification. We are interested in preventing *falls*, understood as any part of the robot other than its feet making contact with the ground. To encode the failure set of all such falls with a simple margin function, we define a small number of critical points \mathbf{p}_c , including the 8 corners of a (tight) 3-D bounding box around the robot’s torso as well as its four knee joints. The safety margin is then:

$$g(x) = \min \left\{ \min_i \{z_{\text{corner}}^i\} - \bar{z}_{\text{corner},g}, \min_i \{z_{\text{knee}}^i\} - \bar{z}_{\text{knee}} \right\},$$

with z_{corner}^i the vertical distance to the ground of the i^{th} robot body corner point and z_{knee}^i the vertical distance to the ground of the i^{th} robot knee point. The target (all-time safe set) is defined as a narrow neighborhood of a static stance with all four feet on the ground and a sufficiently lowered torso, chosen so that the robot is robustly stable with a simple stance controller. The margin function is

$$\begin{aligned} \ell(x) = \min \left\{ \bar{\omega} - |\omega_x|, \bar{\omega} - |\omega_y|, \bar{\omega} - |\omega_z|, \right. \\ \left. \bar{v} - |v_x|, \bar{v} - |v_y|, \bar{v} - |v_z|, \right. \\ \left. \bar{z}_{\text{corner},\ell} - \max_i \{z_{\text{corner}}^i\}, \bar{z}_{\text{foot}} - \max_i \{z_{\text{foot}}^i\} \right\}, \end{aligned}$$

with z_{foot}^i the vertical elevation of the i^{th} robot foot relative to the ground. Table 5 shows the threshold values (\cdot) used to define the safety and target margin functions for quadruped walking.

Uncertainty specification. To account for uncertainty in the deployment conditions as well as general modeling error (or sim-to-real gap), our operational design domain (ODD) includes an external force that may push or pull any point on the robot’s torso in any direction with a maximum magnitude of 50 N

$$d = [F_x, F_y, F_z, p_x^F, p_y^F, p_z^F] \quad (8)$$

where $F = [F_x, F_y, F_z]$ represents the force vector applied at position defined by p_x^F, p_y^F, p_z^F in the body coordinates $p_x^F, p_y^F \in [-0.1, 0.1]$, $p_z^F \in [0, 0.05]$ m. The red arrows in the imagined gameplay of Fig. 2 show examples of learned adversarial disturbance.

C Extended Evaluation

To further demonstrate the strengths of our approach and shed light on its superior scalability to complex robot dynamics, we conduct a comparative analysis of the gameplay performance of the self-play-trained controller and disturbance policies *as training proceeds*. The results in Figures 4 and 5 suggest that the dense temporal difference signal in reach-avoid games plays a determining role in enabling data-efficient learning, while previously proposed safety methods that use reward-based RL with a (sparse) failure indicator consistently require more training episodes before starting to learn meaningfully robust safe control strategies.

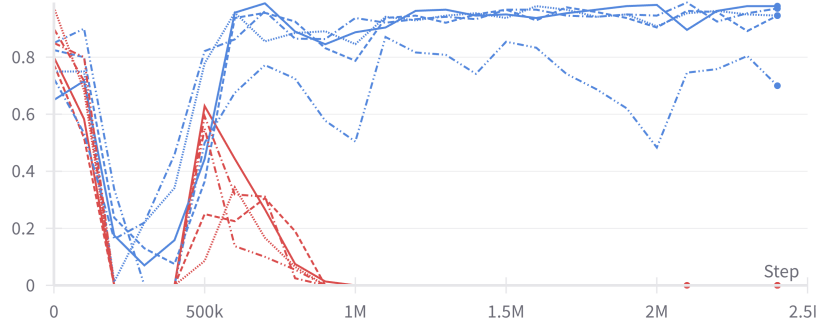


Figure 4: Safe rate achieved by the robot controller against the co-trained adversarial disturbance as the adversarial RL synthesis proceeds under reach-avoid objective (blue) and reward-based objective (red).

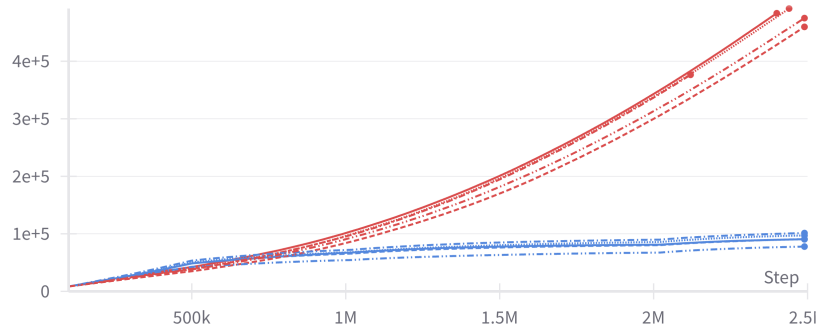


Figure 5: Cumulative safety violation count as the adversarial RL synthesis proceeds under reach-avoid objective (blue) and reward-based objective (red).