

473 This document contains supplementary material for the CoRL 2024 submission 393 "SoloParkour:
 474 Constrained Reinforcement Learning for Visual Locomotion from Privileged Experience". The code
 475 used in the paper will be released upon publication.

476 A Experiment Videos



Figure 6: Solo-12 performing agile locomotion skills in diverse indoor and outdoor environments such as climbing high steps, leaping over gaps, and crawling under obstacles.

477 The supplementary video contains both indoor and outdoor experiments of the system. Figure 6
 478 illustrates these experiments.

479 B Implementation Details

480 B.1 Rewards and Constraints

481 We use the reward function 3 from [13] that measures progress toward a specific direction. To
 482 always have positive rewards, we add a survival bonus of 0.5 at each time step, and, following [5],
 483 we clip total rewards below 0.0.

484 To enforce the constraints, we follow CaT [20] and reformulate the constrained RL problem 1 into
 485 the following RL problem:

$$\max_{\pi} \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \left(\prod_{t'=0}^t \gamma (1 - \delta(s_{t'}, a_{t'})) \right) r(s_t, a_t) \right], \quad (5)$$

486 with termination probabilities $\delta(s_t, a_t)$. Following CaT, we define the termination probabilities as:

$$\delta = \max_{i \in I} p_i^{\max} \text{clip}\left(\frac{c_i^+}{c_i^{\max}}, 0, 1\right), \quad (6)$$

487 where $c_i^+ = \max(0, c_i(s, a))$ is the violation of constraint i , c_i^{\max} is an exponential moving average
 488 of the maximum constraint violation over the last batch of experience collected in the environment,
 489 and p_i^{\max} a hyperparameter that controls the maximum termination probability for the constraint i .

Table 2 lists all the constraints used. Following [20], we separate constraints between hard constraints, where $p_i^{\max} = 1.0$, and soft constraints, where p_i^{\max} increases throughout the course of training, allowing the RL agent to discover agile locomotion during the early stage of training while enforcing more the constraints later on to ensure safe behaviors. To encourage the emergence of a specific locomotion style, some constraints are activated only in specific settings. For instance, the *Stand still* constraints c_{still} are only active when no velocity command is provided, whereas the *Sase orientation* and *Number of foot contacts* are only active on flat terrains and on early terrain levels. We found that rescaling the constraint violations by the square root function $c_i \leftarrow \sqrt{c_i^+}$ helps CaT be less sensitive to extreme values of constraint violations.

Type	Expression	Hard	Cond.
Knee or base collision	$c_{\text{knee/base contact}} = 1_{\text{knee/base contact}}$	✓	×
Foot contact force	$c_{\text{foot contact}_j} = \ f^{\text{foot}_j}\ _2 - f^{\text{lim}}$	✓	×
Foot stumble	$c_{\text{stumble}_j} = \ f_{xy}^{\text{foot}_j}\ _2 - 4 f_z^{\text{foot}_j} $	×	×
Heading	$c_{\text{heading}} = \text{angle}_{\text{base}} - \text{angle}_{\text{cmd}} - \text{angle}^{\text{lim}}$	×	×
Torque	$c_{\text{torque}_k} = \tau_k - \tau^{\text{lim}}$	×	×
Joint velocity	$c_{\text{joint velocity}_k} = \dot{q}_k - \dot{q}^{\text{lim}}$	×	×
Joint acceleration	$c_{\text{joint acceleration}_k} = \ddot{q}_k - \ddot{q}^{\text{lim}}$	×	×
Action rate	$c_{\text{action rate}_k} = \frac{ \Delta q_{t,k}^{\text{des}} - \Delta q_{t-1,k}^{\text{des}} }{dt} - \dot{q}^{\text{des lim}}$	×	×
Joint limits min	$c_{\text{joint}_j^{\text{min}}} = \text{joint}_j^{\text{min}} - \text{joint}_j$	×	×
Joint limits max	$c_{\text{joint}_j^{\text{max}}} = \text{joint}_j - \text{joint}_j^{\text{max}}$	×	×
Foot air time	$c_{\text{air time}_j} = t_{\text{air time}}^{\text{des}} - t_{\text{air time}_j}^{\text{lim}}$	×	×
Base orientation (x-axis)	$c_{\text{ori}} = \text{base ori}_x - \text{base}_x^{\text{lim}}$	×	×
Base orientation	$c_{\text{ori}} = \ \text{base ori}_{xy}\ _2 - \text{base}^{\text{lim}}$	×	✓
Number of foot contacts	$c_{\text{n foot contacts}} = n_{\text{foot contact}} - n_{\text{foot contact}}^{\text{des}} $	×	✓
Stand still	$c_{\text{still}} = \ q - q^*\ _2 - \epsilon_{\text{still}}$	×	✓

Table 2: List of constraints, where *Hard* indicates whether each row corresponds to a hard constraint and *Cond.* indicates whether a constraint is active only under certain conditions.

B.2 Policy Learning

We built our RL algorithm with the CleanRL implementations of PPO and DDPG. During privileged policy learning, we linearly increase the damping parameter (Kd) of the PD controller from 0.05 to 0.2 but keep it fixed at 0.2 during visual policy learning. Indeed, we empirically observed that RL discovers agile skills more easily with lower Kd but policies with higher Kd transfer better to the real Solo-12.

Privileged Policy Learning An MLP parametrizes the privileged policy with hidden dimensions [512, 256, 128] and elu activations. We use PPO [48] with 4096 actors in parallel in simulation. The training procedure is very similar to [5, 13, 20].

Visual Policy Learning The actor processes depth images using a vision neural network consisting of three blocks of a convolution with leaky ReLU activations, followed by max pooling and a linear layer to produce the depth embeddings. Random translation, random noise, and random cutout are applied to the depth images during training. The actor then processes the history of proprioceptive information, actions, and depth embeddings with a one-layer Gated Recurrent Unit [75] (GRU) of hidden size 256. This GRU is followed by a MLP with hidden dimensions [512, 256, 128] and elu activations. The output of the final layer is processed by a tanh activation function and rescaled to produce the 12-dimensional action vector. We used the action bounds observed in the privileged experience buffer to rescale the actions given by the actor. The critic network is parameterized by a

517 MLP with hidden dimensions [512, 256, 128], layer normalization and elu activations to process the
518 privileged state.

519 We generate trajectories from the privileged policy that amounts to 2 million state-action samples
520 and store them in the privileged experience buffer $\mathcal{D}^{\text{priv}}$ whereas online experience is collected by
521 256 actors in parallel into the online replay buffer $\mathcal{D}^{\text{online}}$. We store the constraint violations c_i^+ of
522 both online and privileged experience in their respective replay buffers to recompute the termination
523 probabilities δ on the fly during off-policy learning. Both $\mathcal{D}^{\text{priv}}$ and $\mathcal{D}^{\text{online}}$ store privileged infor-
524 mation at every step and depth image every 5 environment steps. During training, we only give the
525 vision network one image every five timesteps and then replicate the dept latent five times to match
526 the sequence length of the other observations. The online replay buffer stores the GRU hidden latent
527 produced by the online actors whereas the privileged replay buffer stores zeros for these latents. This
528 is done to initialize the first hidden of the DDPG actor correctly during off-policy training.

529 We train the visual policy using a variant of RLPD [21]. We build upon DDPG [70] with an update-
530 to-data ratio of 8 during policy evaluation. We use REDQ [71] with 10 critics and an ensemble of 2
531 random critic targets.

532 C Real Robot Setup

533 We use the Solo-12 quadruped robot for our experiments. We built a custom 3D-printed plastic
534 piece to mount the Intel RealSense D405 stereo camera observing in front of the robot. We use the
535 Python wrapper of librealsense to capture depth images at resolution 424×240 . We resize and crop
536 the images to 48×48 and apply the librealsense postprocessing hole-filling filter. Depth images
537 are preprocessed in a separate thread on a separate CPU as they come, at around 30Hz. The visual
538 policy runs at 50Hz using ONNX and produces target joint angles to torque by a PD controller
539 with stiffness $Kp = 4.0$ and damping $Kd = 0.2$ running at 10KHz. Hence, depth embeddings
540 are updated at a higher frequency at inference than during training. All the computation is done
541 through Python scripts by the onboard Raspberry Pi 5. Velocity commands are sent to the embedded
542 controller via a wireless gamepad.