Reinforcement Learning-Based Control under UDR: From Gym Hopper to Obstacle Avoidance in Webots

Daniele Catalano
Polytechnic of Turin
Turin, Italy
s349472@studenti.polito.it

Samuele Caruso
Polytechnic of Turin
Turin, Italy
s349506@studenti.polito.it

Ramadan Mehmetaj
Polytechnic of Turin
Turin, Italy
s346213@studenti.polito.it

Francesco Dal Cero
Polytechnic of Turin
Turin, Italy
s342631@studenti.polito.it

Abstract

This project investigates the application of Reinforcement Learning (RL) to robotic control, with a focus on sim-to-real transfer—training policies in simulation that generalize to real-world conditions. The first phase used the Gym Hopper environment, where foundational algorithms (REINFORCE, Actor-Critic) and a state-of-the-art method, Proximal Policy Optimization (PPO), were implemented for locomotion. To simulate the reality gap, two custom domains were created: the target with nominal dynamics, and the source with a 30% torso mass reduction. Uniform Domain Randomization (UDR) was applied to the remaining three link masses in the source domain, using manually defined uniform distributions. Mass values were sampled at each episode to promote robustness across dynamic variations. The project then extended to a Webots-based scenario, where a Tesla Model 3 learned obstacle avoidance using LiDAR, GPS, and IMU. PPO and Soft Actor-Critic (SAC) were employed, with UDR applied to obstacle positions and to the vehicle's initial conditions.
(Project git repository)

1 Introduction

Reinforcement Learning (RL) allows agents to learn control policies by interacting with an environment, yet transferring these policies from simulation to the real world remains challenging due to the so-called sim-to-real gap. This gap stems from differences in dynamics, sensor noise, and model inaccuracies between training and deployment settings.

This work addresses this issue using a sim-to-sim setup based on the Gym Hopper environment, introducing a systematic mass perturbation to simulate the domain shift. Policy gradient methods, including REINFORCE and Actor-Critic, are implemented and compared with the state-of-the-art algorithm Proximal Policy Optimization (PPO).

To enhance generalization, Uniform Domain Randomization (UDR) is applied by varying dynamic parameters during training. A project extension explores a realistic autonomous driving task in Webots, where a Tesla vehicle navigates randomized static obstacles using domain randomization. Results demonstrate the advantages of Soft Actor-Critic (SAC) in robustness and adaptation under dynamic conditions.

2 Related Work

Policy gradient methods laid the foundation for RL in robotic control by optimizing parameterized policies $\pi_\theta(a|s)$ to maximize the expected return:

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r(s_t, a_t)\right]$$

Early methods such as **REINFORCE** perform stochastic gradient ascent using Monte Carlo estimates of the return:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\pi\left[\nabla_\theta \log \pi_\theta(a|s) R\right]$$

where R is the cumulative reward. However, this approach suffers from high variance, making learning unstable. A common solution is to introduce a *baseline* $b(s)$, leading to a variance-reduced estimator:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\pi\left[\nabla_\theta \log \pi_\theta(a|s)(R - b(s))\right]$$

where $b(s)$ can be the average return or a learned value function. This adjustment does not change the expected gradient but reduces its variance.

Actor-Critic algorithms further develop this idea by coupling a parameterized policy (actor) with a value function approximator (critic) to estimate the expected return. The critic replaces Monte Carlo returns with bootstrapped estimates, enabling more incremental updates and lower variance.

Building on these foundations, more recent algorithms such as PPO and SAC improve stability and sample efficiency, becoming standard in continuous control tasks.

Proximal Policy Optimization (PPO) [3]: An on-policy method, uses a clipped surrogate objective to constrain updates:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

where $r_t(\theta)$ is the probability ratio between new and old policies, and $\hat{A}_t$ is the estimated advantage.

Soft Actor-Critic (SAC) [8]: An off-policy algorithm, incorporates an entropy term to encourage exploration:

$$J(\pi) = \sum_{t=0}^{T}\mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[r(s_t, a_t) + \alpha\mathcal{H}\big(\pi(\cdot|s_t)\big)\right]$$

while efficiently reusing past experience via a replay buffer.

Domain Randomization (DR) [4]: DR tackles the sim-to-real gap by varying simulation parameters during training. In its uniform variant, physical and sensory parameters are sampled from predefined ranges to expose policies to diverse dynamics and improve robustness under domain shifts.

Simulators: While MuJoCo via Gym is standard for RL benchmarking, the sim-to-real gap hinders policy transfer. Webots, with higher-fidelity physics and sensors, provides a richer testbed for studying generalization. Here, PPO and SAC are extended from Gym's Hopper to a custom Webots [5] Tesla to assess robustness under domain variability.

3 Gym Hopper

3.1 Methodology

The Hopper environment from OpenAI Gym is used to benchmark policy gradient algorithms. It consists of a single-legged robot tasked with hopping forward without falling. To simulate the sim-to-real gap, the source domain modifies the torso mass by reducing it by 30% compared to the target domain.

- **State space:** 11-dimensional continuous vector of joint positions and velocities.
- **Action space:** 3-dimensional continuous torque commands.

3.1.1 Applied Algorithms

Control policies in the Hopper environment were trained using REINFORCE, Actor-Critic, and PPO. These algorithms are described in detail in Section 2.

3.1.2 UDR

UDR is applied by varying environment parameters across training episodes. In Hopper, all link masses except the torso are uniformly randomized within the range 0.5× to 1.5× their nominal values. This exposes the policy to diverse dynamics during training, improving robustness to domain shifts.

3.2 Experiments and Results

This section presents experimental results on the Hopper environment, evaluating the learning performance and stability of REINFORCE, Actor-Critic, and PPO, as well as their robustness to domain variability and sim-to-real transfer under UDR.

3.2.1 REINFORCE vs Actor-Critic

- REINFORCE exhibits high reward variance and slower convergence.
- Actor-Critic achieves more stable learning and lower time to convergence.

3.2.2 PPO Transfer Performance

To improve the performance and stability of PPO in the Hopper environment, a hyperparameter tuning process was performed separately in the source and target domains. Starting from the default Stable-Baselines3 configuration, key hyperparameters were adjusted for each domain. The resulting configurations, selected based on the best results over 50 test episodes, were then applied across the respective experiments. Each agent was trained for 1 million timesteps. The Table 2 compares the baseline and optimized settings.

The bar chart (*Fig. 1*) and box plot (*Fig. 1*) together provide a detailed view of the performance of the PPO source and PPO target policies across the three configurations (source→source, source→target, and target→target).

1. source→source: The policy is trained and tested in the source domain. PPO source achieves a high average return (1600), clearly outperforming PPO default. The box plot shows a tight reward distribution around the median (1595), indicating consistent performance. This serves as the in-domain baseline, confirming that hyperparameter tuning improves stability.

Table 1: Hyperparameters for PPO

Hyperparameter	**Default**	**Source**	**Target**
Learning rate	3.0×10^{-4}	2.5×10^{-4}	2.8×10^{-4}
Discount factor (γ)	0.99	0.995	0.993
GAE	0.95	0.97	0.96
Batch size	64	128	96
Entropy coefficient (α)	0.00	0.01	0.005
Number of steps	2048	4096	3072
Number of epochs	10	15	12
Optimizer	Adam	Adam	Adam

2. source→target: The policy is trained in the source domain and tested in the target domain (sim-to-real transfer). PPO source achieves high average return (1670) with low variance, indicating robust generalization. Although source→target typically suffers from domain shift, here training in the source domain yielded a more robust policy.

3. target→target: The policy is trained and tested in the target domain. PPO target achieves a higher average return (1680) compared to the other configurations. The box plot shows a higher median (1680) with small variance, indicating consistent and optimal performance.

Summary: Hyperparameter tuning in source and target domains enhances performance and stability. As expected, target→target achieves the highest reward, but direct training on the target is rarely feasible, underscoring the importance of transfer methods like DR.

3.2.3 UDR Robustness

The agent was trained with UDR in the source environment using PPO for 1 million timesteps to ensure consistency with the non-UDR model and was

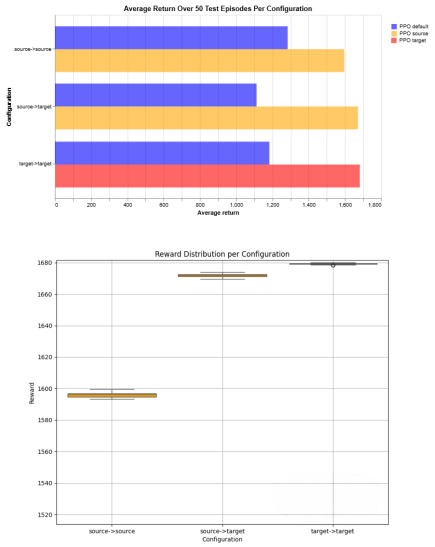

Figure 1: (Top) Average return and (Bottom) reward distribution over 50 episodes per configuration.

then evaluated on both source and target domains to assess transferability. The bar chart (*Fig. 2*) compares average returns with and without UDR across source→source and source→target, while the boxplot (*Fig. 2*) shows the UDR model's reward distribution.

1. source→source: PPO source without UDR achieves a slightly higher average return (1600) than with UDR (1570), showing that domain randomization is not beneficial, in this case, when training and testing on the same domain. Despite this, the boxplot shows a tight reward distribution for UDR, indicating stable and consistent performance.

2. source→target: In the transfer scenario, UDR achieves an average return (1580) slightly lower than PPO source without UDR (1670). This indicates that UDR does not enhance transferability in this context.

Summary: Overall, these results suggest that in this setting UDR does not provide a clear advantage over the standard PPO approach, as it achieves slightly lower average returns, despite exhibiting stable reward distributions. This suggests that the benefits of UDR may depend on the calibration of randomization ranges and the specific dynamics of the transfer scenario.

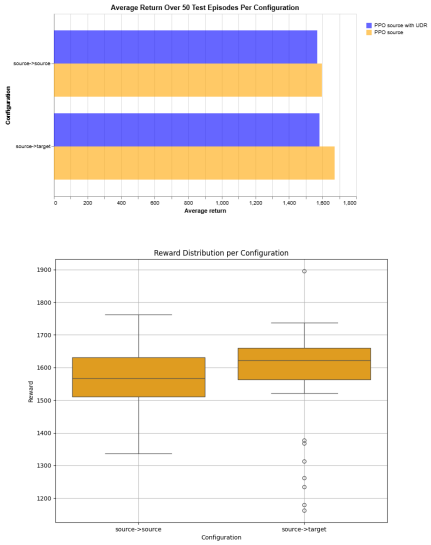

Figure 2: (Top) Average return over 50 episodes per configuration, UDR vs no UDR and (Bottom) reward distribution over 50 test episodes per configuration with UDR.

4 Project Extension: Webots

4.1 Methodology

This work extends Gym Hopper experiments to a more realistic Webots simulation (*Fig. 3*), training a Tesla Model 3 to navigate a 120-meter straight road with three static, diverse obstacles. The controlled yet randomized setup enables robust evaluation of policy generalization and adaptability.

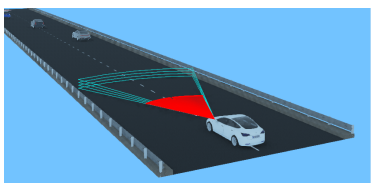

Figure 3: Tesla environment with obstacle placement.

The vehicle uses a minimal sensor suite—front LiDAR (10-beam downsampled), GPS, and IMU—for perception, localization, and instability detection. The reward encourages efficient, goal-directed motion while penalizing collisions, excessive tilt, stagnation, and deviations from optimal dynamics. Episodes end on target reach, collision, flipping, inactivity, or timeout, providing informative feedback for stable learning.

4.1.1 Structure and Functionality of the Car environment

The CustomCarEnv class defines a Webots RL environment where a Tesla Model 3 navigates a static-obstacle course. It provides a Gym-like socket interface and handles simulation logic, including sensors, reward shaping, termination, and scenario variability via Uniform Domain Randomization (UDR).

4.1.2 Observation Space

At each timestep, the environment emits a 20-dimensional observation vector encoding the agent's internal dynamics and environmental perception:

- **[0]**: Normalized average velocity of the rear wheels, based on motor speeds and scaled by the maximum velocity.
- **[1–3]**: Normalized GPS position (x, y, z), with x and y scaled by the road length and half-width, respectively; z is kept unnormalized to retain elevation.
- **[4–6]**: Inertial orientation (roll, pitch, yaw) from the IMU, normalized over $[-\pi, \pi]$ to detect instability or flipping.
- **[7–9]**: Target position in world coordinates, normalized in the same way as GPS to provide a static goal reference.
- **[10–19]**: Front LiDAR readings, downsampled to 10 beams using uniform spacing and normalized by the sensor's maximum range. Infinite values are replaced before normalization.

The LiDAR[6] downsampling strategy is essential for dimensionality reduction and real-time processing. Given a LiDAR scan vector $L \in \mathbb{R}^n$, 10 samples are extracted uniformly:

$$L_{\text{sampled}} = \left\{ L_{i \cdot \lfloor n/10 \rfloor} \,\middle|\, i = 0, \ldots, 9 \right\}$$

Each L_i is clipped at the sensor's maximum range and scaled to the $[0, 1]$ interval.

This observation structure integrates proprioceptive (e.g., velocity, orientation) and exteroceptive (e.g., LiDAR, GPS) data, enabling robust goal-oriented navigation, obstacle avoidance, and stability control.

4.1.3 Action Space

The environment adopts a continuous action space composed of two control variables:

- **[0]**: Normalized velocity command, applied symmetrically to both rear motors of the Tesla vehicle.
- **[1]**: Normalized steering angle, controlling the orientation of the front wheels.

Together, these two actions provide the agent with full low-level control over the vehicle's motion, enabling differential velocity-based propulsion and front-wheel steering.

4.1.4 Reward Function

The reward function promotes safe, efficient, goal-directed navigation, combining progress, safety, and time efficiency components clipped to $[-50, 50]$. Progress is rewarded via distance reduction and a sparse $+100$ bonus at the goal zone. Safety penalties include -50 for collisions (LiDAR < 1.0m), -75 for roll/pitch > 0.2rad, and penalties for idling or proximity (< 1.5m) to obstacles. A mild step penalty discourages delays, while deviations from a 26rad/s reference speed incur soft penalties.

The total reward is:

$$\begin{aligned} r_t = r_{\text{progress}} + r_{\text{goal}} + r_{\text{collision}} + r_{\text{falling}} \\ + r_{\text{blocked}} + r_{\text{speed}} + r_{\text{time}} + r_{\text{proximity}} \end{aligned}$$

clipped as:

$$r_t \leftarrow \text{clip}(r_t, -50, 50)$$

4.1.5 Episode Termination Conditions

An episode terminates under the following conditions:

1. **Target Reached:** Euclidean distance to the goal falls below 3 meters in the (x, y) plane.
2. **Collision:** Minimum front LiDAR reading $<$ 1.0 m while speed $>$ 0.1 rad/s.
3. **Instability:** Absolute roll or pitch exceeds 0.2 rad (IMU-based).
4. **Blocked State:** Movement $<$ 0.01 m and speed $<$ 0.05 rad/s for 50 steps.
5. **Timeout:** Maximum step limit (2000 by default) is reached.

These conditions ensure episodes are informative and enforce safety and task relevance.

4.1.6 Obstacle Randomization using UDR

To improve generalization, UDR is applied at each episode's start (*Fig. 4*). While road layout and task structure remain fixed, obstacle type and placement, along with the vehicle's initial lateral position and heading, are randomized. Obstacles are drawn from a predefined pool and placed in longitudinal segments with safe lateral offsets, exposing the agent to diverse spatial configurations and fostering robust policy learning.

4.1.7 Training Infrastructure and Parallelization

To manage the computational demands of deep RL, PPO and SAC were optimized for efficiency and trained on Google Cloud[2] (22 vCPUs, 44 GB RAM). PPO ensures stable on-policy updates but limits sample reuse, while SAC leverages off-policy experience replay and parallelism[7]. To accelerate data collection, ten parallel Webots instances were launched. Weights & Biases (W&B) handled logging and experiment tracking, enabling scalable and reproducible training.

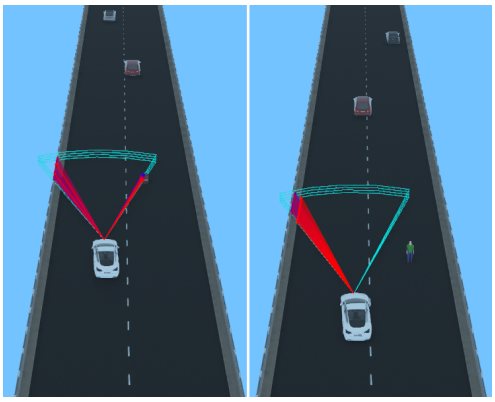

Figure 4: UDR across different episodes.

4.2 Experiments and Results

Experiments in the Webots simulator assess the performance of PPO and SAC in a realistic autonomous driving task. The focus is on generalization under domain randomization and the effect of environment variability.

4.2.1 Model selection

Following the theoretical overview, PPO and SAC were empirically evaluated in Webots to assess sample efficiency, robustness, and adaptability under domain randomization. Metrics such as success rate, stability, rewards, and task completion guided the selection of the final policy. The next sections detail the experimental setup, logging, and comparative results.

4.2.2 Experimental Setup

Both PPO and SAC were trained for 2 million timesteps:

- **Phase 1 (0–1M):** Training without UDR
- **Phase 2 (1M–2M):** Training with UDR, starting from Phase 1 checkpoints

Training was parallelized over 10 environments on a cloud infrastructure, with each instance covering 200,000 steps: 100,000 in a static setup (no UDR)

and 100,000 with UDR to encourage generalization. Performance metrics were logged via Weights & Biases (W&B) from a representative instance, and key hyperparameters[1] are reported in Table 2.

Table 2: Hyperparameters for PPO and SAC

Hyperparameter	**PPO**	**SAC**
Learning rate	3×10^{-4}	3×10^{-4}
Discount factor (γ)	0.99	0.99
Batch size	512	512
Replay buffer size	—	1×10^{6}
Entropy coefficient (α)	—	Auto-tuned
Clipping range (ϵ)	0.2	—
Optimizer	Adam	Adam

4.2.3 Success and Generalization

Success rate analysis (*Fig. 5*) shows SAC's superior learning and generalization. It quickly reaches consistent success in the static phase (0–100k steps) and adapts rapidly after UDR is enabled. In contrast, PPO struggles even in static conditions and fails to cope with domain randomization.

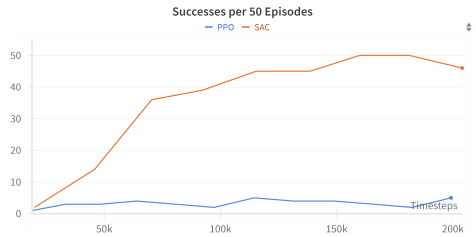

Figure 5: Successes per 50 episodes for PPO and SAC.

4.2.4 Policy Stability and Adaptation

SAC demonstrates greater policy stability, as shown by the Trajectory Variance plot (*Fig. 6*). Although SAC explores extensively early on, the variance decreases as learning progresses. Upon UDR activation at 100k steps, the variance increases moderately, indicating adaptation rather than instability. PPO, in contrast, shows erratic variance throughout, worsening under UDR.

The SAC entropy coefficient (*Fig. 7*) further confirms this adaptability, dropping during static training and briefly increasing post-UDR to enable exploration—a behavior absent in PPO.

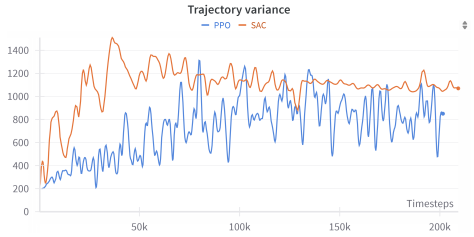

Figure 6: Trajectory variance for PPO and SAC.

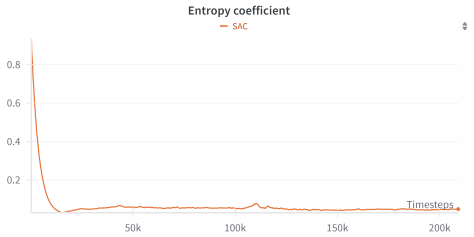

Figure 7: Entropy coefficient evolution for SAC (auto-tuned).

4.2.5 Reward and Efficiency

Reward metrics underline SAC's advantage: the Mean Episode Reward (*Fig. 8*) rises quickly and remains high under UDR, while Distance Covered (*Fig. 9*) shows consistent track completion. SAC's higher Mean Episode Length (*Fig. 10*) reflects successful goal completion, unlike PPO's shorter, failure-prone episodes.

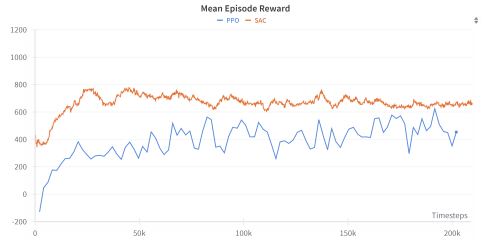

Figure 8: Mean episode reward over time for PPO and SAC.

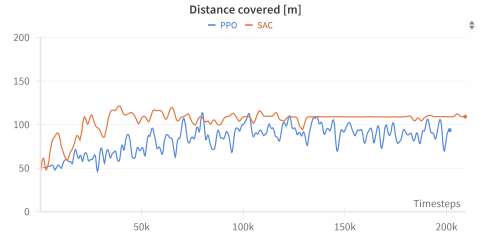

Figure 9: Distance covered per episode for PPO and SAC.

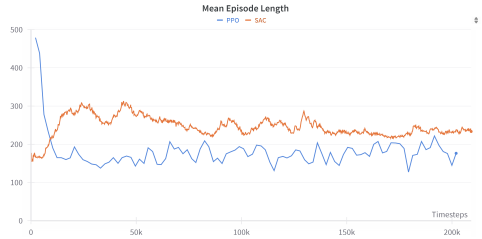

Figure 10: Mean episode length for PPO and SAC.

4.2.6 Assessment

Empirical results show SAC to be more sample-efficient and better at generalizing under environmental randomization. Its off-policy design and entropy tuning support robust adaptation to UDR, while PPO proved less stable and efficient. SAC was therefore selected as the preferred model.

5 Conclusion

This work investigated RL for robotic control with a focus on sim-to-real transfer. PPO performed well in Gym Hopper, while SAC was more efficient and adaptable in Webots under domain variability, underscoring the role of algorithm design and domain randomization in robust policy generalization. Looking ahead, future extensions could explore adaptive domain randomization techniques that dynamically adjust environment variability based on the agent's performance, accelerating generalization and robustness. Additional enhancements such as multi-agent scenarios, dynamic obstacles, complex road topologies, and enhanced sensor models with noise injection or visual inputs from camera sensors would further increase environmental richness and foster policies capable of meeting the challenges of real-world deployment.

References

[1] Theresa Eimer, Marius Lindauer, and Roberta Raileanu. Hyperparameters in reinforcement learning and how to tune them. 2023. 7

[2] Lindita Nebiu Hyseni and Afërdita Ibrahimi. Comparison of the cloud computing platforms provided by amazon and google. 2017. 6

[3] Prafulla Dhariwal Alec Radford Oleg Klimov John Schulman, Filip Wolski. Proximal policy optimization algorithms. 2017. 2

[4] Mingjun Ma, Haoran Li, Guangzheng Hu, Shasha Liu, and Dongbin Zhao. Comparison of different domain randomization methods for policy transfer in reinforcement learning. 2023. 2

[5] Olivier Michel. WebotsTM: Professional mobile robot simulation. 2004. 2

[6] Benjamin Quito and Larbi Esmahi. Compare and contrast lidar and non-lidar technology in an autonomous vehicle: Developing a safety framework. 2023. 5

[7] Yeqiong Shi and Lu Wang. Highly parallel implementation of machine learning algorithms based on reconfigurable structures. 2023. 6

[8] Pieter Abbeel Sergey Levine Tuomas Haarnoja, Aurick Zhou. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 2018. 2