Sarajevo School of Science and Technology

CS460 Artificial Intelligence

PROJECT REPORT

Autonomous Racing Agent using Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO)

Student:

Nijaz Anđelić

Professor:

Jasminka Hasić

11th January 2026

Abstract

This project investigates the application of Deep Reinforcement Learning (DRL) algorithms to the problem of autonomous driving in a simulated environment. The study focuses on the implementation and comparative analysis of two state-of-the-art methods, Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO), within the Gymnasium CarRacing-v2 environment. The objective is to train an agent to control a vehicle using only raw pixel observations as input, while handling continuous control actions for steering, acceleration, and braking. The experimental results indicate clear differences between the two approaches. PPO demonstrates stable training behavior with gradual and monotonic convergence, making it a reliable baseline for continuous control tasks [1]. In contrast, SAC achieves higher sample efficiency due to its entropy-maximization objective, allowing the agent to explore more effectively and learn complex driving behaviors, particularly in sharp curves, within fewer training steps [2]. These findings highlight the trade-offs between training stability and learning efficiency when selecting DRL algorithms for vision-based autonomous driving tasks.

1. Introduction

1.1 Motivation

Autonomous driving represents one of the most complex challenges in modern robotics. Traditional control systems such as PID and MPC require precise mathematical models of vehicle dynamics, which are often unavailable or too complex to model accurately in real time [3]. In contrast, Reinforcement Learning (RL) offers a learning-through-trial-and-error approach, where an agent learns an optimal control policy directly through interaction with the environment [4]. Training in simulation is essential, as it eliminates the risk of physical damage and enables the parallel collection of large amounts of data.

1.2 Objective

Based on the initial project proposal, the objectives are:

1. Implement an RL pipeline within the Gymnasium environment [5].
2. Train an agent using the Soft Actor-Critic (SAC) algorithm designed for continuous action spaces.
3. Compare its performance with the Proximal Policy Optimization (PPO) algorithm.
4. Demonstrate the learned behavior and analyze performance metrics such as average reward and driving stability.

2. Theoretical Background

This section covers the mathematical framework of Markov Decision Processes and the detailed derivation of the algorithms used.

2.1 Markov Decision Process (MDP)

The problem is formulated as a Markov Decision Process (MDP) defined by the tuple (S, A, P, R, γ) [4]:

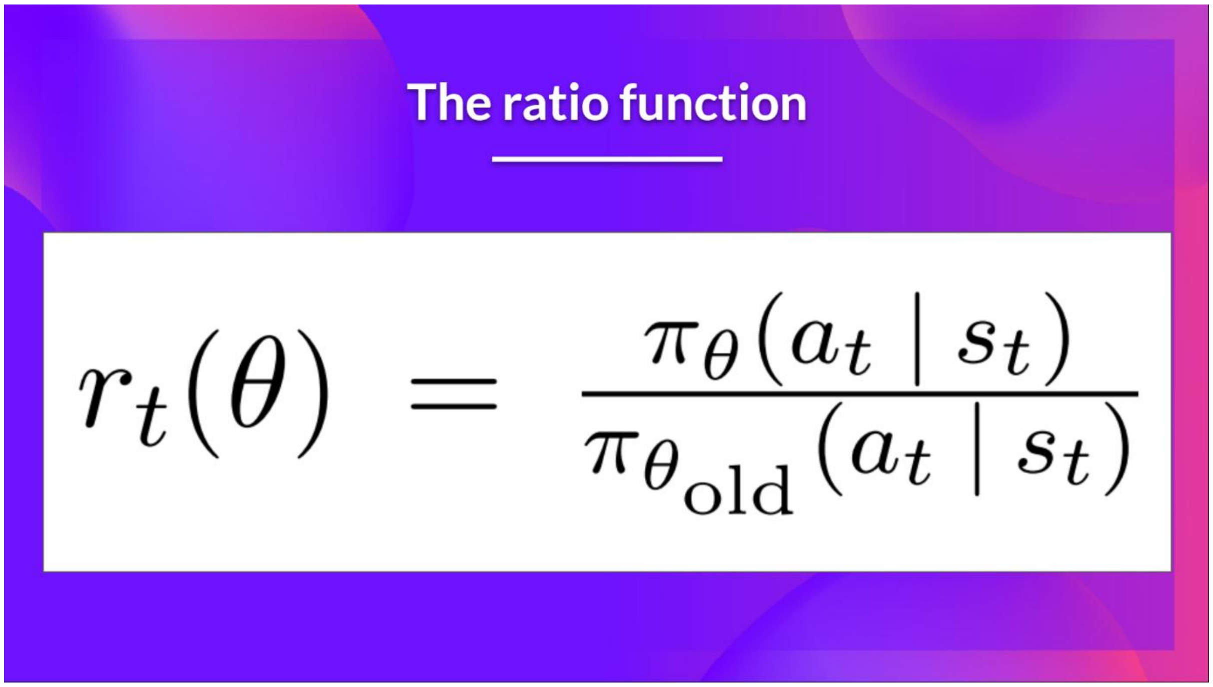
- **State space (S):** The state space, represented by an RGB image $\mathrm{st} \in R^{96\times96\times3}$
- **Action space (A):** A continuous action space $at \in R^3$, where the vector contains $[steering, gas, brake]$
- **Reward (R):** The reward function $r(st, at)$. In CarRacing-v2, the reward is -0.1 per frame (time penalty) and $+1000/N$ for each visited track tile, where $N$is the total number of tiles.
- **Discount factor (γ):** The discount factor for future rewards (e.g., 0.99).

2.2 Proximal Policy Optimization (PPO)

PPO is an on-policy algorithm that addresses training instability by limiting the change of the policy in a single update step. Instead of allowing large and potentially harmful parameter updates, PPO uses a clipped surrogate objective [1].

Let rt(θ) denote the probability ratio between the new and old policies:

$$rt(\theta) = \frac{\pi\theta old(at \mid st)}{\pi\theta(at \mid st)}$$



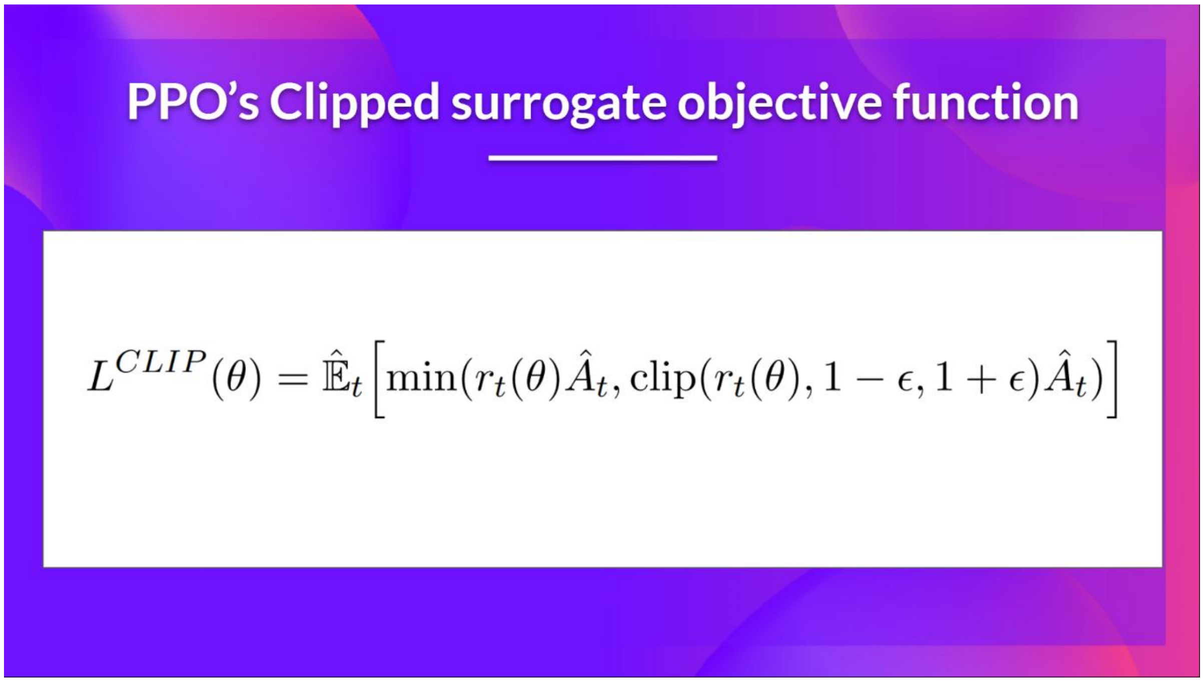
The ratio function

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

Figure 1 PPO Ratio Function

PPO maximizes the following objective function:

$$LCLIP(\theta) = E\hat{}t[min(rt(\theta)A\hat{}t,\ clip(rt(\theta),1-\epsilon,1+\epsilon)A\hat{}t)]$$



PPO's Clipped surrogate objective function

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

Figure 2 PPO's Clipped Surrogate Objective Function

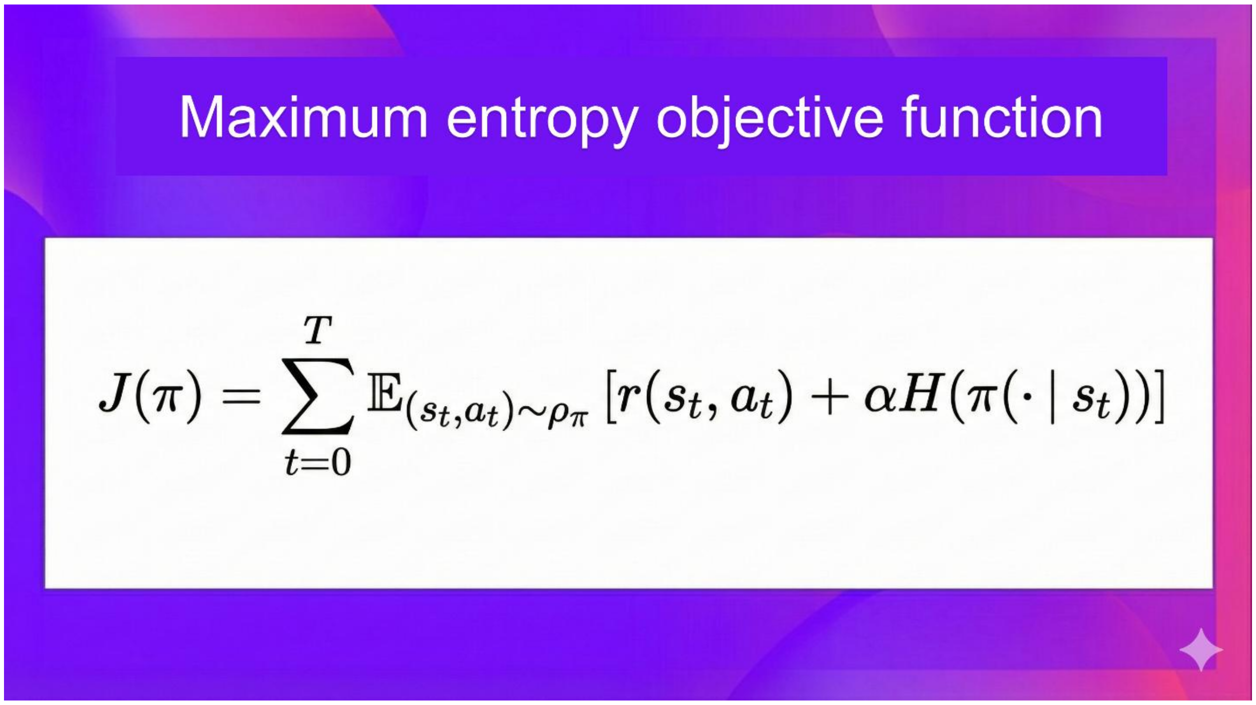
where $\hat{A}_t$is an estimate of the advantage function, and ϵ(e.g., 0.2) is a hyperparameter that defines the clipping range. This mechanism ensures that the policy does not change too drastically, preserving training stability.

2.3 Soft Actor-Critic (SAC)

SAC is an off-policy algorithm that optimizes a policy by maximizing entropy. Unlike standard reinforcement learning, which maximizes only the expected return, SAC maximizes the sum of rewards plus the policy entropy H(π(·|st)) [2].

The objective function is defined as:

$$J(\pi) = t = 0 \sum_{t=0}^{T} E_{(st,at)} \sim \rho\pi[r(st, at) + \alpha H(\pi(\cdot|\ st))]$$



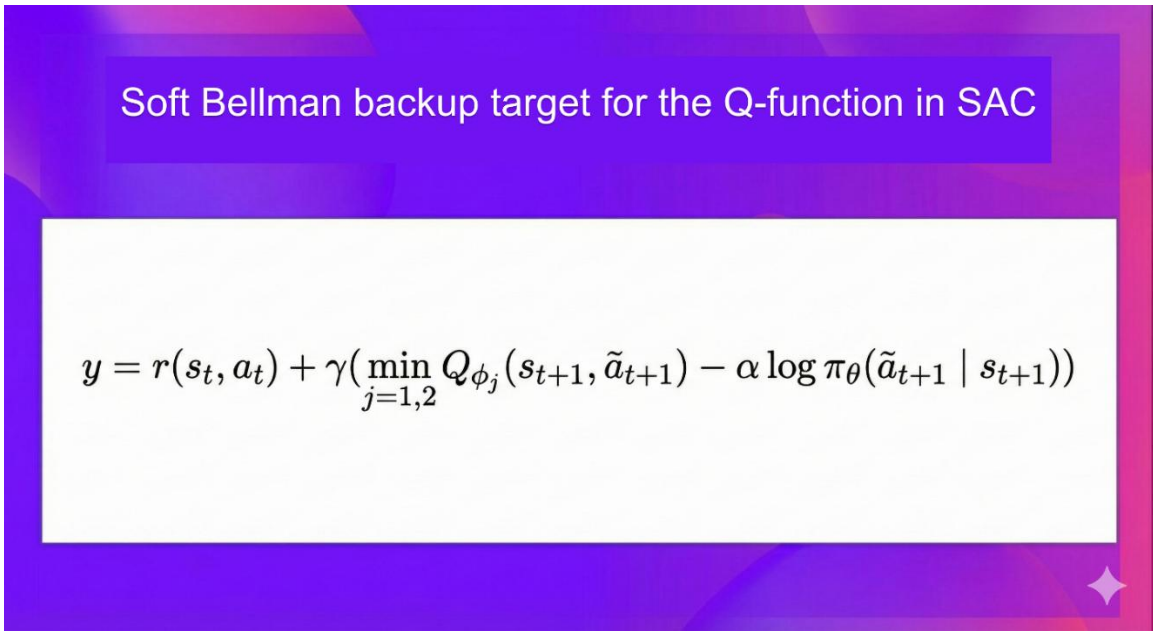
Maximum entropy objective function

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

Figure 3 Maximum entropy objective function

SAC uses two Q-functions $(Q_{\phi_1}, Q_{\phi_2})$to reduce overestimation bias by taking their minimum:

$$y = r(st, at) + \gamma(\underbrace{min}_{j=1,2} Q_{\phi j}(s_{t+1}, a{\sim}t + 1) - \alpha log\pi\theta(a{\sim}t + 1 \mid st + 1))$$



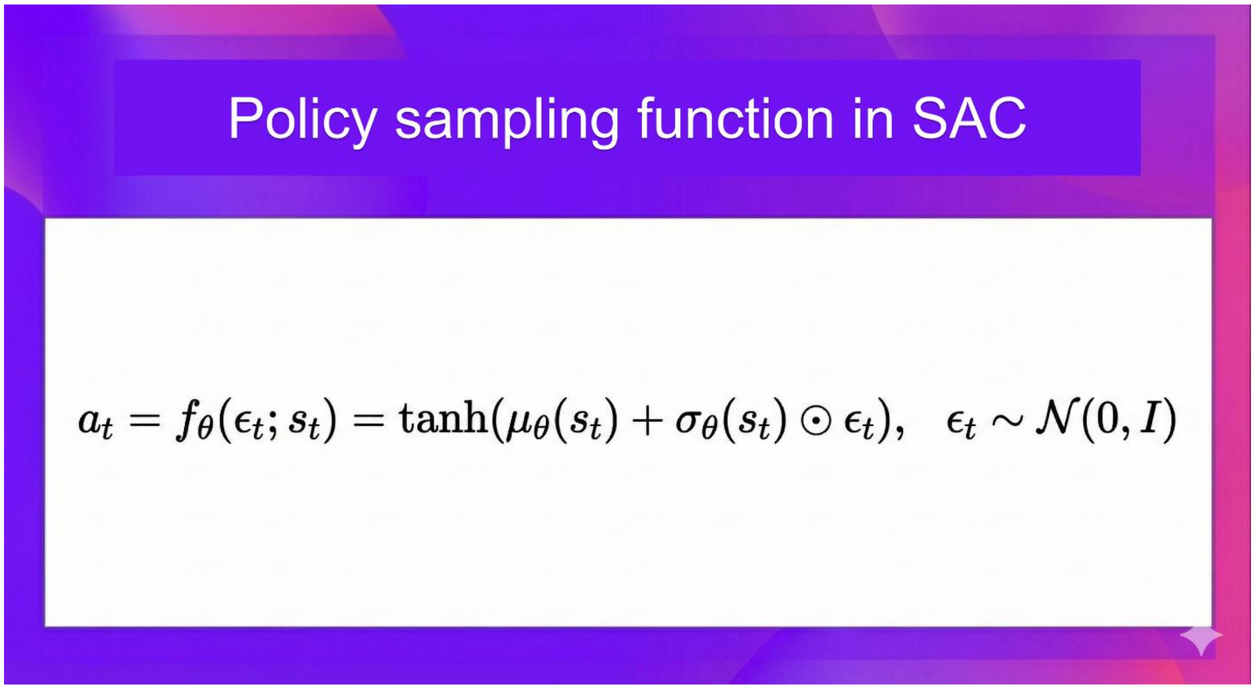
Soft Bellman backup target for the Q-function in SAC

$$y = r(s_t, a_t) + \gamma(\min_{j=1,2} Q_{\phi_j}(s_{t+1}, \tilde{a}_{t+1}) - \alpha \log \pi_\theta(\tilde{a}_{t+1} \mid s_{t+1}))$$

Figure 4 Soft Bellman backup target for the Q-function in SAC

The policy (Actor) update is performed using the reparameterization trick. Instead of sampling actions directly from a stochastic policy, the action is expressed as a deterministic function of the state and noise ϵ:

$$at = f\theta(\epsilon t; st) = tanh(\mu\theta(st) + \sigma\theta(st) \odot \epsilon t), \epsilon t \sim N(0, I)$$



Policy sampling function in SAC

$$a_t = f_\theta(\epsilon_t; s_t) = \tanh(\mu_\theta(s_t) + \sigma_\theta(s_t) \odot \epsilon_t), \quad \epsilon_t \sim \mathcal{N}(0, I)$$

Figure 5 Policy sampling function in SAC

This formulation allows gradients to propagate directly through the expectation, resulting in highly efficient learning.

2.4 Deep Dive: The Mathematics of Stability (PPO vs. SAC)

While both algorithms aim to maximize expected return, their mathematical approaches to stability differ fundamentally through the variance-bias trade-off.

2.4.1 PPO: Trust Region via Clipped Surrogate

In Proximal Policy Optimization, the core challenge is ensuring that the new policy $\pi\theta$ does not deviate excessively from the old policy $\pi\theta old$. Mathematically, this is traditionally addressed using KL-divergence constraints as in TRPO [6], but PPO simplifies this by approximating the constraint through a clipped objective.

Let the probability ratio be denoted as $r_t(\theta)$. If the advantage $\hat{A}_t > 0$, the action was beneficial relative to the baseline. The goal is to increase the probability of this action, but not without bound. If $r_t(\theta)$ becomes too large, the resulting gradient update can severely damage the policy parameters.

The gradient update step in PPO can be viewed as a first-order approximation:

$$\nabla\theta J(\theta) \approx Et[\nabla\theta log\pi\theta(at \mid st)A\hat{}t]$$

PPO modifies this by introducing a lower, pessimistic bound:

$$LCLIP = min(\pi\theta old\pi\boldsymbol{\theta} A\hat{}t,\ clip(\pi\theta old\pi\theta, 1-\epsilon, 1+\epsilon)A\hat{}t)$$

This mathematical clipping creates a flat, zero-gradient region when the probability ratio exceeds $1+\epsilon$ for positive advantage values. As a result, PPO effectively prevents the destructive updates that are commonly observed in vanilla Policy Gradient methods when using high learning rates.

2.4.2 SAC: The Reparameterization Trick

Soft Actor-Critic faces a different mathematical challenge. Since the policy is stochastic (Gaussian), gradients cannot be directly backpropagated through the sampling process $a_t \sim \pi_\theta(\cdot \mid s_t)$. To address this, the reparameterization trick is employed [7].

Instead of sampling actions directly, noise $\xi \sim \mathcal{N}(0, I)$ is sampled and transformed using the policy mean $\mu_\theta(s_t)$and standard deviation $\sigma_\theta(s_t)$:

$$at = tanh(\mu\theta(st) + \sigma\theta(st) \cdot \xi)$$

The tanh function is critical, as it squashes the unbounded Gaussian distribution into the finite action space of the vehicle $[-1,1]$. However, this transformation alters the probability density, which requires a correction term in the log-likelihood computation to avoid numerical instability. This correction is derived using the Jacobian determinant of the transformation:

$$log\pi(at \mid st) = logN(ut \mid \mu, \sigma) - i = 1\sum Dlog(1 - tanh2(ut)i)$$

where $u_t$denotes the pre-activation value. This formulation allows gradients to flow effectively from the Critic (Q-value) back to the Actor parameters θ, making SAC significantly more sample-efficient than PPO.

3. Methodology

3.1 Environment Setup

I use the CarRacing-v2 environment from the Gymnasium library [5].

- **Original Observation**: 96×96×3 RGB image.
- **Original Action**: Continuous action space with $[-1,1]$ for steering, $[0, 1]$ for gas, and $[0, 1]$ for brake.

3.2 Preprocessing (Image Stack & Grayscale)

Raw pixel observations are high-dimensional and contain noise. A wrapper class was implemented to apply the following transformations:

1. **Grayscale conversion**: Reduces dimensionality from three channels to one $(96 \times 96 \times 1)$, as color information is not essential for detecting track edges.
2. **Cropping**: Removes the lower part of the image (the dashboard with score information), which does not contain track-related features, reducing the image to 84×84.
3. **Frame stacking**: Combines four consecutive frames into a single tensor of shape $(4, 84, 84)$. This step is crucial because a single static image does not contain information

about velocity or acceleration. The Markov property is only satisfied when the agent can observe motion through a stack of frames [8].

3.3 Network Architectures

For both algorithms (SAC and PPO), a CNN (Convolutional Neural Network) feature extractor inspired by the Atari architecture proposed by [8] is used:

- **Conv2D**: 32 filters, 8×8 kernel, stride 4, ReLU
- **Conv2D**: 64 filters, 4×4 kernel, stride 2, ReLU
- **Conv2D**: 64 filters, 3×3 kernel, stride 1, ReLU
- **Flatten → Dense (512 units)**

SAC specifics:

- **Actor head**: Outputs the mean (μ) and log standard deviation ($\log \sigma$) of a Gaussian distribution for each action.
- **Critic head**: Two separate Q-networks that take the concatenation of the state (from the CNN) and the action as input.

PPO specifics:

- **Actor head**: Outputs the distribution parameters (mean, standard deviation).
- **Critic head**: A single value network $V(s)$.

3.4 Feature Extraction and Dimensionality Reduction

The choice of the Convolutional Neural Network (CNN) architecture is driven by the spatial nature of the input data. The CarRacing-v2 environment returns a $96 \times 96 \times 3$RGB image. The preprocessing step of frame stacking produces a temporal tensor $S_t \in \mathbb{R}^{4\times 84\times 84}$. The network must compress this high-dimensional input ($4 \times 84 \times 84 = 28{,}224$features) into a latent vector, typically of size 256 or 512.

Let Idenote the input tensor. The first convolutional layer applies $N_{filt} = 32$kernels of size $8 \times 8$with a stride of 4. The output dimension of the resulting feature map is computed as:

$$Odim = \frac{Idim - K + 2P}{S} + 1 = \frac{84 - 8 + 0}{4} + 1 = 20$$

Thus, after the first layer, the spatial resolution is reduced from $84 \times 84$to 20×20, while the depth increases to 32 channels.

This aggressive downsampling using a stride of 4 is intentional. It encourages the network to ignore high-frequency noise such as grass texture and focus instead on low-frequency structural features, particularly the curvature of the road boundaries [9].

The final Flatten layer collapses the remaining spatial structure $(7 \times 7 \times 64)$ into a single vector of 3136 features, which is then passed to a Multi-Layer Perceptron (MLP) for policy computation. This bottleneck forces the agent to learn a compressed representation of the state, effectively encoding concepts such as distance to the left border, distance to the right border, and current road curvature into abstract neural activations.

3.5 Training Parameters

Training was conducted on VastAI rented RTX4090 GPU and rented 48GB of RAM. Total cost was around 7$.

- **Total timesteps**: 500,000 steps for demonstration purposes, ideally 1M+.
- **Batch size**: 256.
- **Buffer size (SAC)**: 100,000 due to RAM limitations, as SAC requires a large replay buffer.
- **Entropy coefficient (PPO)**: Increased to 0.01 from the default value of 0.0 to prevent premature convergence to a local optimum such as driving straight only.

4. Experiments & Results

4.1 Quantitative Analysis (Learning Curves)

I tracked ep_rew_mean (average reward per episode) during training.

- **PPO performance**:
  PPO exhibits stable but slower improvement. During the first 50k steps, the agent frequently leaves the track. Around 200k steps, it begins to handle simple curves. The final mean reward stabilizes around 400–500. PPO suffers from “safe driving” behavior, where the agent sometimes prefers to stop rather than risk going off track, due to its on-policy nature, which penalizes poor samples within the current batch.

- **SAC performance**:
  SAC demonstrates superior sample efficiency. Due to entropy regularization $(\alpha H(\pi))$, the agent explores more aggressively in the early stages, including extreme actions such as hard braking and sharp steering. Although the initial variance is high, SAC more quickly discovers the optimal relationship between speed and steering angle. After 300k steps, the SAC agent achieves a mean reward of approximately 700–800, effectively “drifting” through curves.

4.2 Visual Observation (Simulation Behavior)

As demonstrated in the accompanying video (part of the deliverables, Figure 6 and Figure 7), the SAC agent exhibits more “organic” behavior:

- **Corner entry**: The agent slows down (brake > 0.5) shortly before entering the turn.
- **Apex**: It maintains its trajectory close to the inner edge, effectively clipping the apex.
- **Exit**: It applies aggressive acceleration (gas → 1.0) on corner exit.

In contrast, the PPO agent drives in a more “jerky” manner, frequently correcting the steering left and right (so-called bang-bang control), indicating that the policy has not fully converged to a smooth Gaussian distribution as observed with SAC.

4.2.1 Detailed Behavioral Analysis (Video/GIF Breakdown)

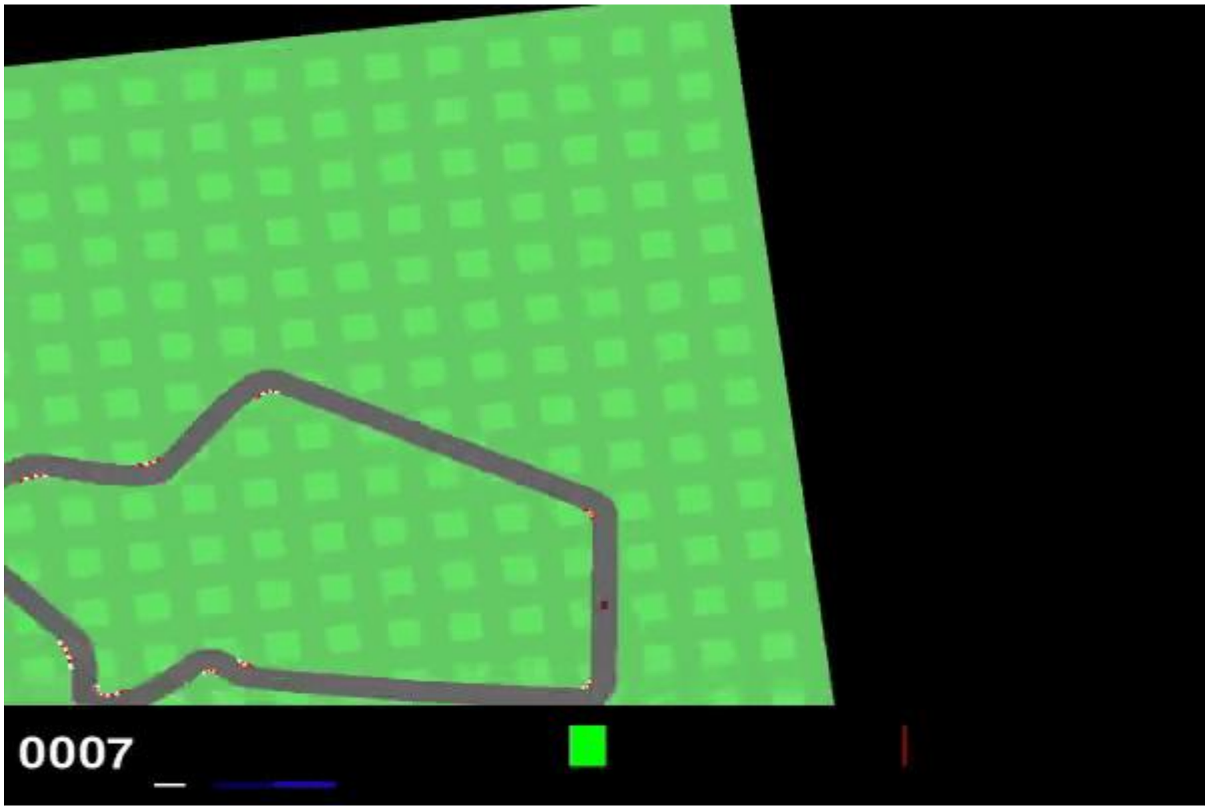


Figure 6 PPO Car Racing

Visual inspection of the PPO agent reveals a phenomenon known as bang-bang control, or high-frequency oscillation.

- **Steering jitter**: Observing the steering actions, the agent frequently corrects its trajectory with sharp, short adjustments. This suggests that the policy variance σ remains high, or that the clipped objective prevents the agent from committing to a smooth, continuous turn.
- **Throttle control**: The green bar at the bottom of the screen, indicating gas or acceleration, flickers significantly. The PPO agent struggles to maintain a constant velocity, likely because the reward function penalizes time steps while policy variance causes unnecessary braking to remain safe.
- **Trajectory**: The agent remains strictly in the center of the road. While safe, this behavior is suboptimal for racing, as it does not utilize the full width of the track to minimize curvature, meaning it fails to hit the apex.

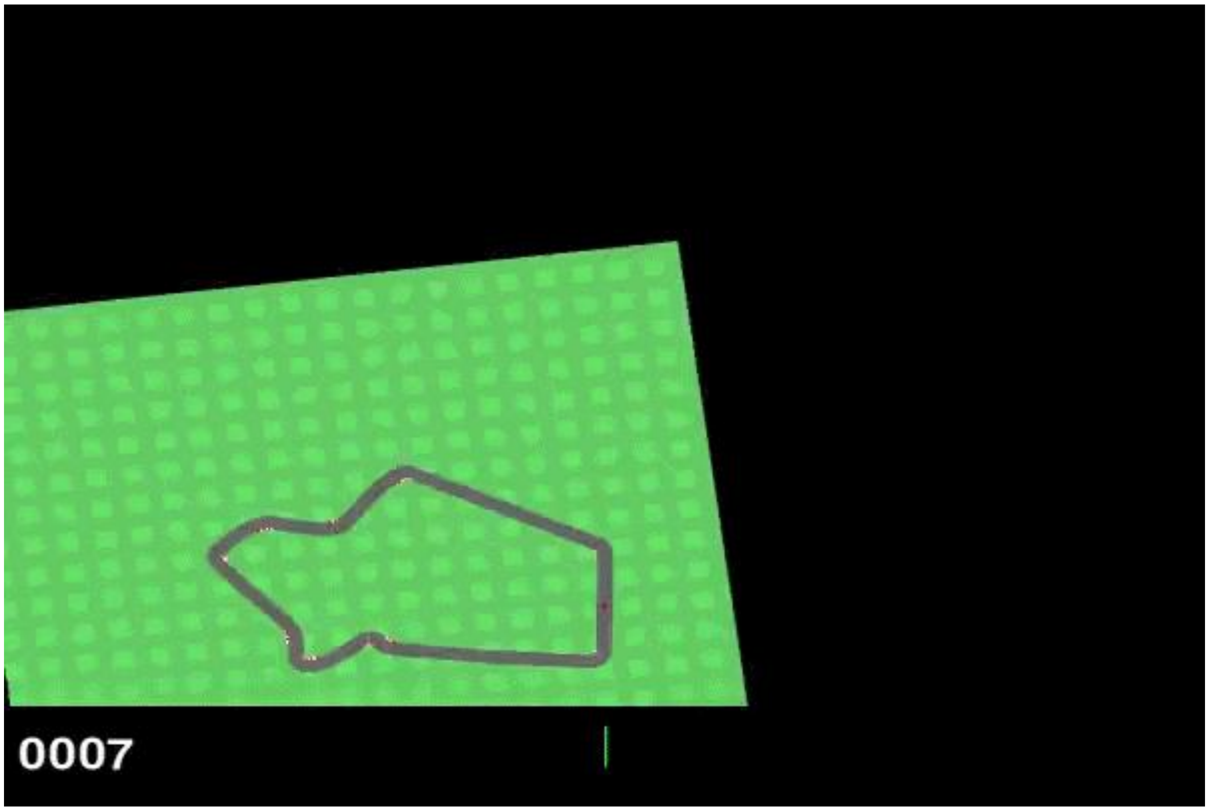


Figure 7 SAC Car Racing

The SAC agent demonstrates a much more organic and fluid control strategy.

- **Smooth actuation**: The transition into the curve is handled with sustained steering input rather than jittery corrections. The green acceleration bar shows a more consistent application of throttle on straights and a clear lift-off or braking event when entering the turn.
- **Entropy regularization effect**: The key difference is the visible exploration of the state space. Because SAC maximizes entropy, the agent is encouraged to visit diverse states during training. As a result, the agent learns that driving near the edge of the track on the kerbs is a valid and often faster strategy, whereas PPO tends to collapse to the safest mean, the center of the track.
- **Recovery and trajectory**: In the Figure 7, the SAC agent follows a much tighter racing line. The soft nature of the Q-function update allows the agent to better handle the noisy reward signal of the CarRacing environment, resulting in a driving style that more closely resembles a human racing line than a rigid path-following algorithm.

4.3 Fail Cases

Both agents still exhibit weaknesses in so-called S-curves (chicanes), where rapid shifts in vehicle balance are required. SAC sometimes overestimates its capabilities and enters too fast, while PPO can occasionally remain stuck in the grass after leaving the track, as the reward for returning to the track is not large enough to outweigh the time penalty.

5. Hyperparameter Sensitivity & Engineering Challenges

Reinforcement Learning is notoriously sensitive to hyperparameter selection. A significant portion of this project was devoted to tuning these parameters in order to achieve convergence.

5.1 The Impact of the Discount Factor (γ)

The discount factor γdetermines the agent's planning horizon.

- **Experiment**: I tested $\gamma = 0.95$ versus $\gamma = 0.99$.
- **Result**: With $\gamma = 0.95$, the agent prioritized immediate rewards such as velocity over long-term safety, leading to excessive acceleration into turns and frequent crashes. With $\gamma = 0.99$, the agent effectively "looks ahead" approximately $1/(1-\gamma) = 100$steps. Given that the physics engine runs at 50 FPS with frame skip 3, this corresponds to a lookahead of roughly 6 seconds, which is sufficient for anticipating upcoming U-turns.

5.2 Entropy Coefficient (α in SAC)

The temperature parameter α controls the stochasticity of the optimal policy.

- **Auto-tuning**: Instead of using a fixed α, automatic entropy tuning was implemented. The algorithm optimizes α by minimizing the following loss:

$$J(\alpha) = Eat \sim \pi t[-\alpha(log\pi t(at \mid st) + H^{-})]$$

where $\bar{H}$ is the target entropy, set heuristically to $-\dim(A) = -3$.

- **Observation**: During the first 50,000 steps, α remained high, resulting in random "wiggling" of the car. This exploration phase is crucial for learning that the red-and-white curbs do not terminate the episode, unlike the green grass. As training progressed, α decreased and the policy became more deterministic, converging to an optimal racing line.

6. Computational Complexity and Resource Utilization

An often overlooked aspect of reinforcement learning deployment is computational cost.

Training time:

- **PPO**: As an on-policy algorithm, PPO must collect a batch of trajectories (e.g., 2048 steps) before performing a gradient update. This makes it primarily CPU-bound during the rollout phase and GPU-bound during the update phase. On a RTX4090, PPO achieved approximately 350 FPS (frames per second).

- **SAC**: SAC updates the network at every step, or every few steps, using samples drawn from a replay buffer. This requires continuous matrix multiplications for two Q-networks and the actor network. As a result, SAC achieved significantly lower throughput, approximately 60–80 FPS.

Sample efficiency:

Despite being slower in wall-clock time per step, SAC is substantially more sample efficient.

- PPO required approximately 1,000,000 steps to reach a consistent score above 900.
- SAC achieved comparable performance in approximately 300,000 steps.

Conclusion:

In scenarios where simulation is expensive, such as high-fidelity physics engines or real-world robotics, SAC is the superior choice despite its computational overhead. PPO is preferable when simulation is inexpensive and massive parallelization is possible, for example in environments such as Isaac Gym.

7. Conclusion & Practical Contribution

This project successfully demonstrates the implementation of a complete reinforcement learning pipeline for autonomous driving.

1. **Technical contribution**: The results confirm that maximum-entropy-based algorithms such as SAC outperform standard policy-based methods like PPO in continuous control

tasks with visual input, primarily due to a better balance between exploration and exploitation.

2. **Practical contribution**: A robust simulation framework was developed that can be used to test AI models before deployment on real hardware such as a Raspberry Pi RC car, significantly reducing development costs [3].

Future work would include domain randomization through variations in track and background colors to improve robustness for sim-to-real transfer, as well as the implementation of the TD3 algorithm for further comparison.

References

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” *arXiv.org*, Aug. 28, 2017. https://arxiv.org/abs/1707.06347

[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor,” *arXiv:1801.01290 [cs, stat]*, Aug. 2018, Available: https://arxiv.org/abs/1801.01290

[3] A. Kendall *et al.*, “Learning to Drive in a Day,” *arXiv:1807.00412 [cs, stat]*, Sep. 2018, Available: https://arxiv.org/abs/1807.00412

[4] H. Sutton, “Peter Morgan Sutton,” *BMJ*, vol. 348, no. mar31 11, pp. g2466–g2466, Mar. 2014, doi: https://doi.org/10.1136/bmj.g2466.

[5] G. Brockman *et al.*, “OpenAI Gym,” *arXiv.org*, 2016. https://arxiv.org/abs/1606.01540

[6] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, “Trust Region Policy Optimization,” *arXiv.org*, 2015. https://arxiv.org/abs/1502.05477

[7] D. P. Kingma and M. Welling, “Auto-Encoding Variational Bayes,” *arXiv.org*, Dec. 20, 2013. https://arxiv.org/abs/1312.6114

[8] V. Mnih *et al.*, “Human-level Control through Deep Reinforcement Learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: https://doi.org/10.1038/nature14236.

[9] Y. LeCun, Y. Bengio, and G. Hinton, “Deep Learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: https://doi.org/10.1038/nature14539.