CS5180
Reinforcement Learning and Sequential Decision Making
Final Project Report

# Learning autonomous race driving with action mapping-based reinforcement learning

TEAM DETAILS:
Saikiran Juttu
Shayak Bhadraray

## Abstract

Autonomous race driving presents a challenging control problem due to the need to operate vehicles at their physical limits while maintaining safety and adherence to complex nonlinear dynamics. Traditional reinforcement learning (RL) approaches often struggle with state-dependent constraints, such as those imposed by tire-road friction, leading to either unsafe behavior or overly conservative policies. This report attempts to explore other existing novel RL-based frameworks that integrate an Action Mapping (AM) mechanism to explicitly handle these constraints during training and execution. The AM mechanism transforms unconstrained virtual actions from the policy network into real control inputs that respect the vehicle's dynamic friction limits. A numerical approximation method is proposed to implement the mapping, enabling compatibility with continuous control algorithms like DDPG, SAC, and TD3. Extensive experiments conducted in a custom race simulation environment thus demonstrate the success of the discussed AM-RL method when compared to conventional RL approaches. Moreover, the AM mechanism facilitates policy generalization across varying friction conditions, enhancing the robustness and adaptability of autonomous race control systems.

# Contents

# Chapter 1

# Introduction

## 1.1 Problem Statement

Autonomous race driving is a highly challenging control task that demands vehicles to operate at the very edge of their dynamic and physical limits. Unlike traditional autonomous driving tasks such as urban navigation or highway cruising, race driving involves high-speed maneuvers with complex nonlinear vehicle dynamics, making it difficult to model and control. Reinforcement Learning (RL) offers a promising framework for handling such complex control problems by learning policies through interaction with the environment. However, standard RL approaches often use penalty-based techniques to enforce constraints, which leads to conservative policies that underutilize the available control limits. This limits both safety and performance in safety-critical scenarios like autonomous racing.

To address this, a recent line of work has introduced Action Mapping (AM) as a way to incorporate hard constraints into RL policies by transforming unconstrained actions into safe control commands. This report builds upon this approach and extends it by applying AM not only with TD3 (as done in the original paper) but also with other actor-critic RL algorithms like DDPG and SAC. Our goal is to explore the generalizability and effectiveness of AM across different RL paradigms.
The central challenge in autonomous race driving lies in generating control policies that:

- Maximize speed while minimizing lap times,
- Respect vehicle dynamics and physical limitations (e.g., tire friction),
- Avoid safety violations like skidding, drifting, or off-track driving,
- Generalize across different friction conditions and race scenarios.

Traditional control methods and even standard RL approaches fail to fully address these challenges. Penalty-based constraint handling leads to overly cautious behavior, while hard constraints are difficult to encode directly into policy learning. This necessitates a more flexible, yet robust mechanism to incorporate constraints without degrading policy performance.

## 1.2 Approach

### 1.2.1 Problem Formulation

Our approach is inspired by the recent work on Action Mapping-based RL (AM-RL)[1]. The main idea is to decouple policy learning from constraint enforcement by allowing the neural network to output unconstrained actions, which are then projected into the feasible control space using a mapping function that ensures constraint satisfaction.

We implement and compare AM with multiple RL algorithms — TD3, DDPG, and SAC to evaluate how each algorithm performs under the same constraint-aware setup in a custom-built race simulation environment.

The autonomous race driving task is formulated as a Markov Decision Process (MDP), represented by the tuple:

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$$

where:

- $\mathcal{S}$: State space, representing the observation of the environment.
- $\mathcal{A}$: Action space, representing the control inputs applied to the car.
- $\mathcal{P}$: State transition dynamics determined by the vehicle model.
- $\mathcal{R}$: Reward function that provides scalar feedback for each state-action pair.
- $\gamma$: Discount factor for future rewards.

**Observation Space ($\mathcal{S}$):** The observation at each timestep includes the following:

- $v_x$: Longitudinal velocity of the vehicle.
- $\omega$: Yaw rate (rotational velocity).
- $\delta$: Front wheel steering angle.
- $d_c$: Lateral distance from the vehicle to the centerline of the track.
- $\phi$: Heading angle error with respect to the track.
- $\mathbf{V}_{\text{FO}}$: A set of $N_{\text{FO}}$ forward observation vectors indicating the curvature of the track ahead.

Thus, the full state vector is:

$$s_t = [v_x, \omega, \delta, d_c, \phi, \mathbf{V}_{\text{FO}}] \in R^n$$

**Action Space ($\mathcal{A}$):** The action vector at time $t$ consists of:

$$a_t = [a_x, a_y] \in [-1, 1]^2$$

- $a_x$: Virtual longitudinal control action (throttle/brake).
- $a_y$: Virtual lateral control action (steering).

These virtual actions are later mapped to real, constraint-safe control inputs $u_t = [u_x, u_y]$ using the Action Mapping mechanism (see Section 1.2.2).

**Reward Function ($\mathcal{R}$):** The reward at each time step encourages high speed in the track direction and penalizes unsafe behavior. It is defined as:

$$r_t = v_x \cos(\phi) + R_{\text{out}} + R_{\text{wrong}} + R_{\text{friction}}$$

Where:

- $v_x \cos(\phi)$: Speed component in the direction of the track.
- $R_{\text{out}} = -100$: Penalty for going off-track.
- $R_{\text{wrong}} = -100$: Penalty for wrong-way driving.
- $R_{\text{friction}} = -100$: Penalty for violating tire-road friction constraints (only for baseline RL agents without AM).

### 1.2.2 RL with Action Mapping - AM

Reinforcement Learning (RL) algorithms for continuous control typically rely on actor-critic frameworks where a policy network (actor) and a value function (critic) are jointly learned to optimize long-term reward. In our study, we explore the effectiveness of the Action Mapping (AM) mechanism using three prominent actor-critic algorithms: Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG (TD3), and Soft Actor-Critic (SAC). Below, we briefly describe each algorithm:

**Deep Deterministic Policy Gradient (DDPG):** DDPG[2] is a model-free, off-policy RL algorithm designed for continuous action spaces. It consists of two neural networks:

- **Actor**: Outputs deterministic actions given a state.
- **Critic**: Estimates the Q-value for a given state-action pair.

The actor directly maps states to actions without sampling from a probability distribution, which makes it suitable for deterministic control tasks like race driving. DDPG employs techniques such as experience replay and target networks to stabilize training.

**Twin Delayed DDPG (TD3):** TD3[3] builds upon DDPG and introduces improvements to reduce overestimation bias and improve training stability:

- **Twin Critic Networks**: Two Q-networks are used, and the smaller Q-value is taken to mitigate overestimation.
- **Delayed Policy Updates**: The actor is updated less frequently than the critic, which leads to more stable training.
- **Target Policy Smoothing**: Noise is added to the target action to regularize value estimation and encourage exploration.

TD3 is considered more robust than DDPG and is used as the baseline in the original AM-RL paper.

**Soft Actor-Critic (SAC):** SAC[4] is an off-policy RL algorithm that introduces entropy maximization to encourage exploration. Unlike DDPG and TD3, SAC is a stochastic actor-critic method:

- The actor outputs a probability distribution over actions rather than deterministic actions.
- A temperature parameter $\alpha$ controls the trade-off between reward maximization and entropy.

This leads to improved exploration and robustness, especially in environments with sparse or deceptive rewards.

**Action Mapping (AM):** Traditional RL agents handle constraints using soft penalties in the reward function, which does not guarantee constraint satisfaction and often results in conservative behavior. Action Mapping (AM) addresses this by introducing a two-stage action generation pipeline:

- **Virtual Policy** $\pi_V(s)$: Outputs unconstrained, normalized actions $a_t = [a_x, a_y] \in [-1, 1]^2$.
- **Real Policy** $\pi_R(s)$: Uses a mapping function to convert $a_t$ into constraint-safe actions $u_t = [u_x, u_y]$.

A key safety constraint considered in this problem is the **friction circle constraint**, which ensures that the combined longitudinal and lateral forces remain within tire-road limits:

$$\sqrt{F_x^2 + F_y^2} \leq \mu_{\max} F_z$$

Since this constraint is highly state-dependent, AM uses a numerical approximation approach:

1. Discretize the vehicle state space (e.g., $v_x$, $\delta$, direction $\theta$).
2. Pre-compute the maximum admissible action norm $\rho_{\max}$ that satisfies the friction constraint.
3. During policy execution, scale the virtual action by $\rho_{\max}$ using multidimensional interpolation.

AM allows the actor to learn freely in the unconstrained space while ensuring that the final control inputs respect physical limitations at every step. This mechanism can be integrated with any actor-critic RL algorithm without modifying the core training loop, making it a flexible and powerful approach for safety-critical tasks like autonomous racing.

# Chapter 2

# Algorithms and Implementation

## 2.1 Environment Setup

We developed a custom racing environment using Python and Matplotlib to simulate a simplified but dynamic racetrack. The environment follows the structure of OpenAI Gym and includes the following key components:

- **Vehicle Dynamics**: A single-track bicycle model is implemented with realistic steering and velocity control. Dynamics are integrated using the Runge-Kutta 4 (RK4) method to ensure smooth trajectory updates.
- **Track Generation**: A 2D racetrack is plotted using Matplotlib, with variable curvature and lane width to test the agent's ability to generalize under non-trivial conditions.
- **Observation Space**: Includes vehicle speed ($v_x$), yaw rate ($\omega$), steering angle ($\delta$), lateral distance to the centerline ($d_c$), heading error ($\phi$), and $N$ look-ahead curvature points ($\mathbf{V}_{\text{FO}}$).
- **Reward Function**: Encourages high-speed progression in the correct direction while penalizing off-track behavior, wrong-way driving, and (for baseline methods) friction constraint violations.
- **Action Mapping Integration**: All policies output normalized virtual actions, which are then mapped to real, safe control inputs using the AM module described in Section 1.2.2.

The environment is designed to be modular so that any off-policy actor-critic algorithm can be tested with or without Action Mapping.

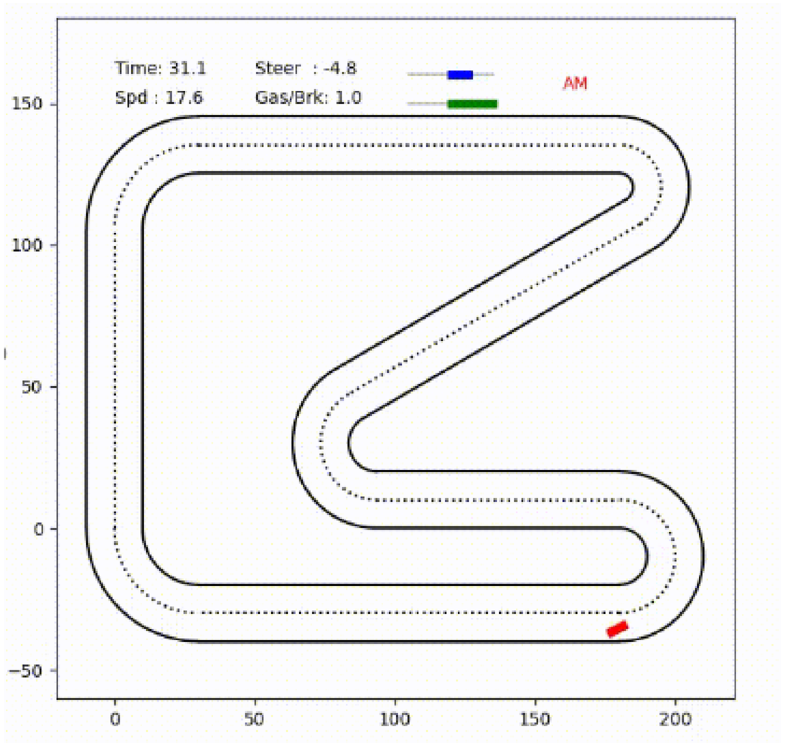


Figure 2.1: Custom Race Environment

## 2.2 Deep Deterministic Policy Gradient (DDPG)

We implemented the DDPG algorithm using PyTorch. The architecture consists of:

- **Actor Network**: A fully connected neural network that takes the state vector as input and outputs a continuous action in the virtual space $[-1, 1]^2$.
- **Critic Network**: Takes both the state and action as input and estimates the Q-value of the given pair.

**Key features in our DDPG implementation:**

- **Experience Replay Buffer**: Stores tuples of $(s, a, r, s')$ for sampling mini-batches and stabilizing learning.
- **Target Networks**: Slow-moving copies of the actor and critic networks updated using a soft update rule:
$$\theta_{\text{target}} \leftarrow \tau\theta + (1-\tau)\theta_{\text{target}}, \quad \tau \ll 1$$
- **Exploration Noise**: We used Ornstein-Uhlenbeck (OU) noise to promote exploration in the continuous action space.
- **Action Mapping**: During training and evaluation, the virtual actions from the actor are passed through the AM module to produce real actions satisfying the tire-road friction constraint.

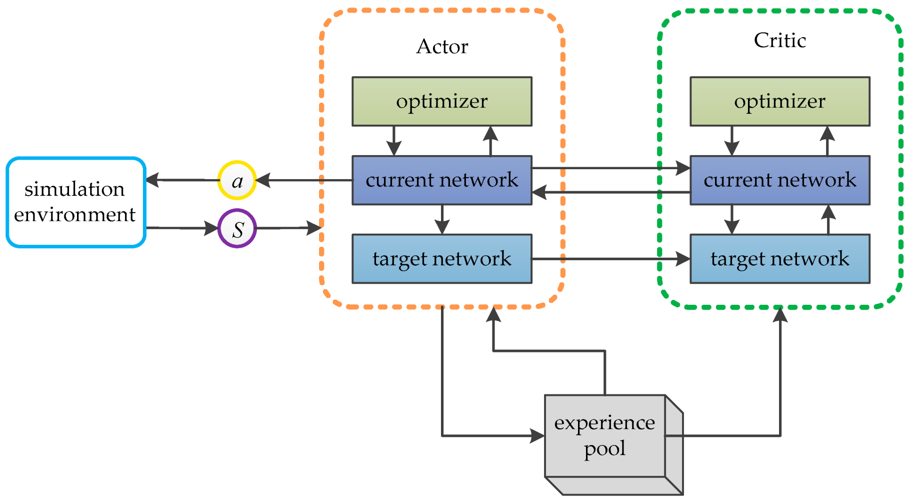


Figure 2.2: DDPG System Architecture

## 2.3 Soft Actor-Critic (SAC)

SAC was implemented as a separate module using the entropy-regularized actor-critic framework. Unlike DDPG, SAC uses a stochastic policy and optimizes both reward and policy entropy.

**SAC Implementation Details:**

- **Stochastic Actor**: Outputs the parameters of a Gaussian distribution (mean and std) over actions, from which real actions are sampled.
- **Critic Network(s)**: Similar to TD3, SAC uses two Q-value networks and a value network to improve stability.
- **Entropy Coefficient** ($\alpha$): Controls the trade-off between exploitation and exploration. We used automated entropy tuning as described in the original SAC paper.
- **Replay Buffer and Target Networks**: As in DDPG, SAC uses experience replay and soft target updates.
- **Action Mapping Integration**: After sampling the virtual action from the Gaussian policy, the action is mapped using the AM mechanism to ensure constraint satisfaction.

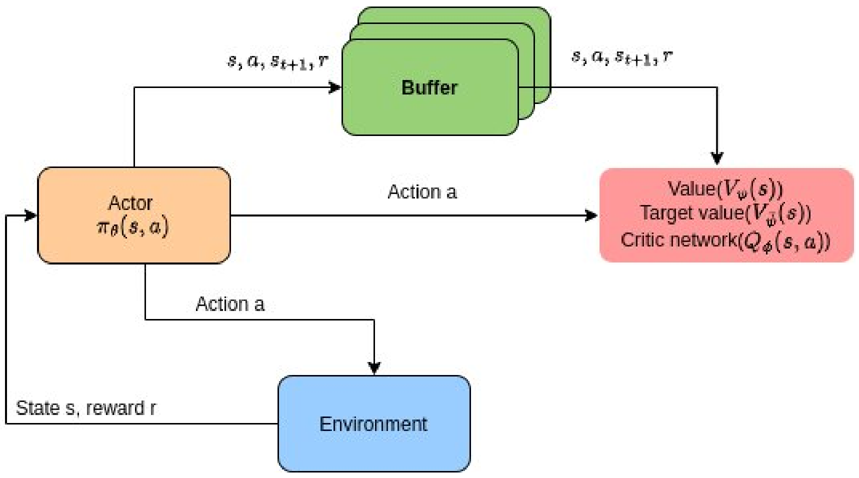


Figure 2.3: SAC System Architecture

# Chapter 3

# Results and Discussion

## 3.1 Training Procedure

Each RL agent was trained in the custom-built racetrack environment discussed in Chapter 2. The training process included the following key components:

- **Episodes and Timesteps**: Agents were trained for 30000 episodes, each with a fixed maximum number of timesteps.
- **Action Mapping**: All virtual actions produced by the actor networks were passed through the AM module to ensure they adhered to the friction constraints.
- **Evaluation Intervals**: Intermediate evaluations were performed at regular intervals manually as the models were being generated to visualize trajectory behavior and assess the performance.
- **Logging**: Rewards, constraint violations, and off-track occurrences were logged to assess agent performance over time.
- **Checkpointing and retraining**: Several iterations of re-training were done by changing the number of episodes and batch size for the experience replay buffer on the basis of the previously trained models, and a checkpointing mechanism was also implemented to resume training from the previously stored model.

## 3.2 DDPG

The DDPG agent with AM demonstrated effective learning of stable policies capable of navigating the track safely. The Key results obtained were:

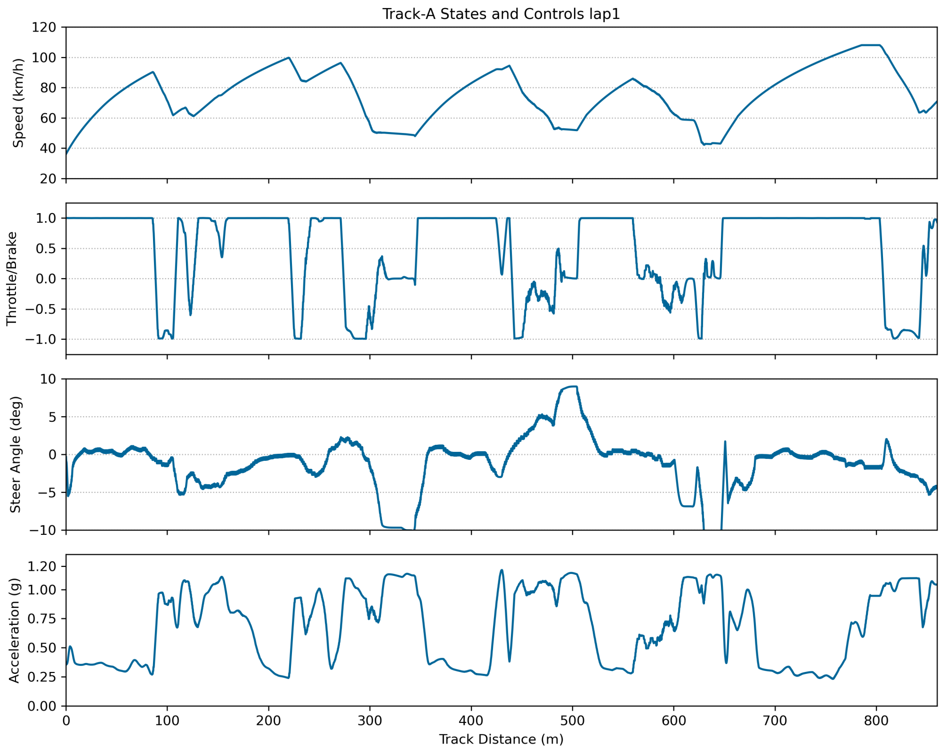


Figure 3.1: Action Mapping outputs for Lap 1

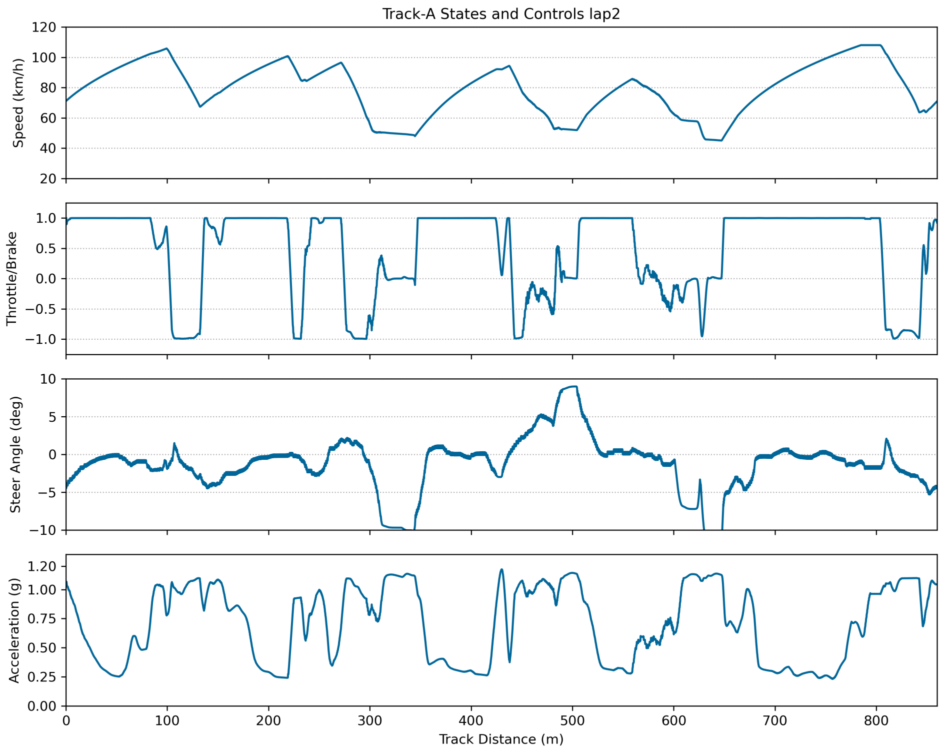


Figure 3.2: Action Mapping outputs for Lap 2

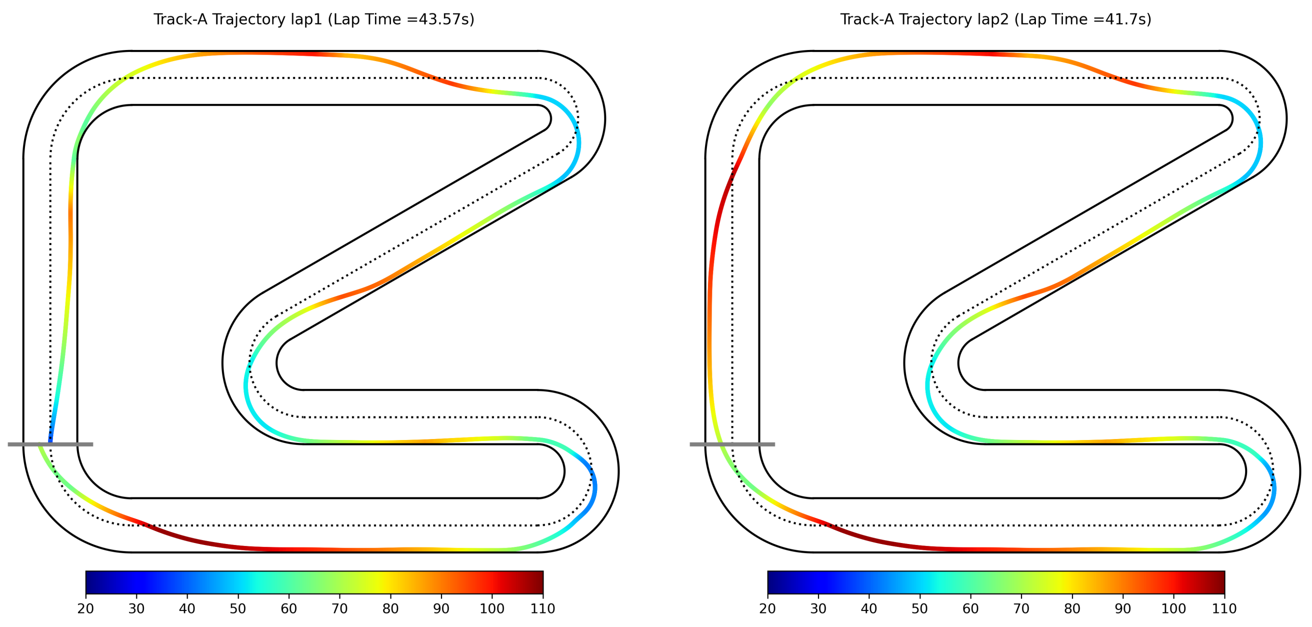


Figure 3.3: Lap 1 and 2 completion by DDPG

## 3.3 SAC

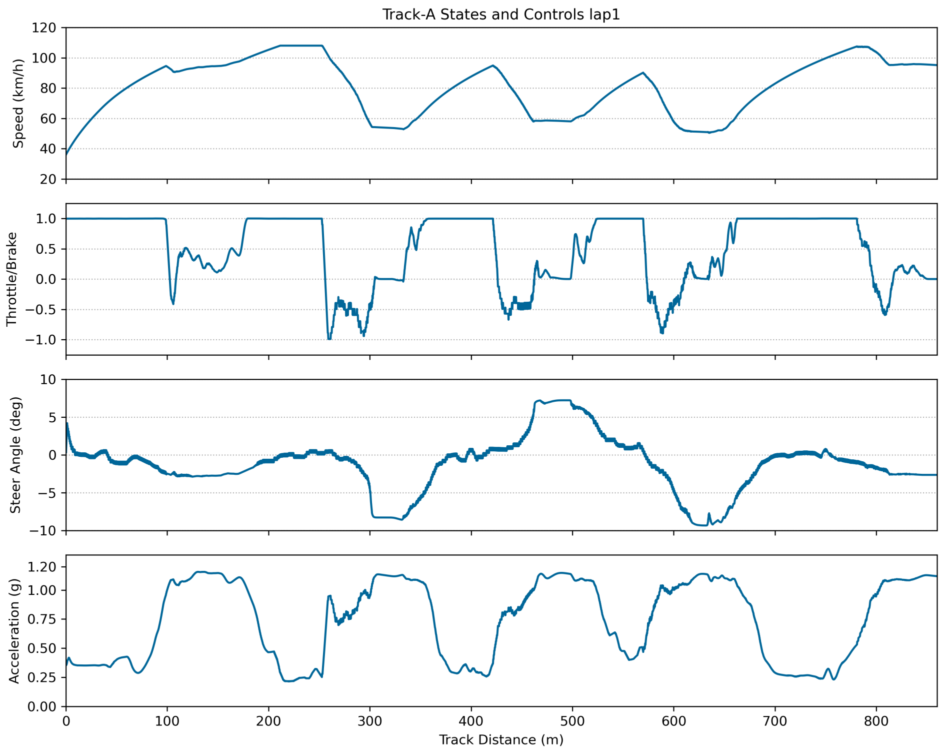
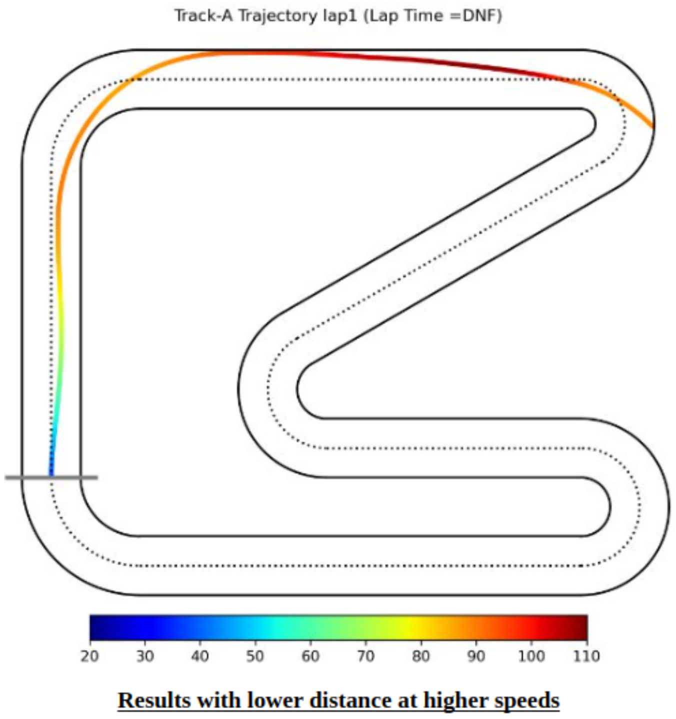


Figure 3.4: Action Mapping outputs for Lap 1



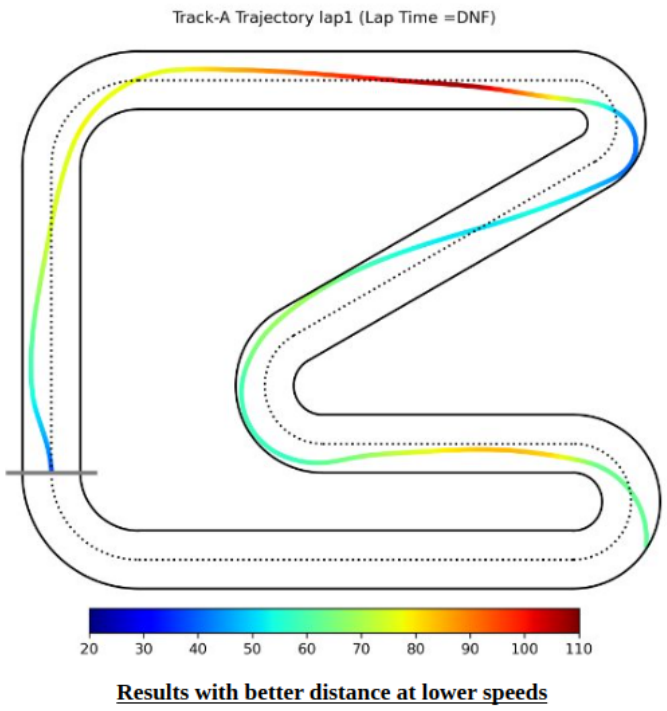


Figure 3.5: Intermediated results of Lap 1 by SAC

## Reward Function Evaluation and Modification

Despite extended training over several thousand episodes using the original reward structure:

$$r_t = v_x \cos(\phi) + R_{\text{out}} + R_{\text{wrong}} + R_{\text{friction}},$$

the SAC agent consistently converged to a similar reward value, regardless of the speed profile adopted. As shown in **Figure 3.5**, the agent exhibited two dominant behaviors: either covering shorter distances at higher speeds or achieving better path adherence at lower speeds. However, both policies resulted in similar overall rewards, suggesting the agent had converged to a local optimum with no significant gain in performance.

To encourage better exploration and lap completion, we introduced a modified reward function that included a progress term. The new reward is defined as:

$$r_t = v_x \cos(\phi) + R_{\text{out}} + R_{\text{wrong}} + R_{\text{friction}} + R_{\text{progress}},$$

This formulation reinforces forward movement along the track (via the `progress` term), while maintaining directional alignment using the $\cos(\phi)$ component. However, as illustrated in **Figure 3.6**, the agent exploited this reward by executing a small turning maneuver near the starting point, returning to the same location and misinterpreting it as lap progress. As a result, it accumulated a high reward without making meaningful forward progress, effectively converging to a **sub-optimal policy**.

This outcome highlights the importance of carefully balancing reward terms; even well-intentioned modifications can lead to reward hacking, where the agent maximizes reward without solving the intended task.

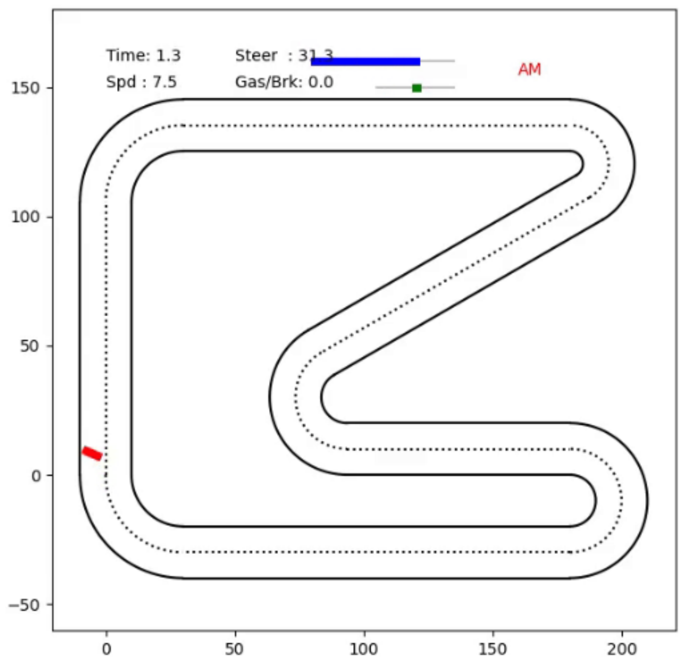


Figure 3.6: SAC stuck in Sub Optimal Policy

### Training Limitations

It is important to note that our experiments were conducted under limited computational resources, which constrained the total training time available. As a result, we were only able to run SAC training for up to **10,000 episodes** per configuration. While this number is sufficient to observe convergence trends and preliminary policy behavior, it may not be enough to fully explore the policy space or escape local optima—especially in complex, high-dimensional continuous control environments.

Given that SAC is an off-policy algorithm with strong asymptotic performance, it is possible that with significantly more training—on the order of tens of thousands of episodes—the agent might eventually learn to overcome the sub-optimal turning behavior and complete full laps successfully. However, due to time and hardware constraints, we were unable to evaluate this possibility conclusively. Future work using extended training budgets or more sample-efficient algorithms may be able to address this limitation and provide deeper insight into long-term policy improvement.

## 3.4 Comparison with TD3

The original research paper used TD3 as the base algorithm for Action Mapping. While SAC could not be implemented by us for a fair comparison, we performed the following comparison between DDPG and TD3. Based on the reported results and our comparative implementation, the following observations were made:

- **Performance:** TD3-AM achieved the best lap times and lowest number of constraint violations among all evaluated methods. DDPG-AM came close in performance.
- **Stability:** TD3's use of twin critics and delayed updates provided more stability during training compared to DDPG.

- **Ease of Tuning:** DDPG required fewer tuning efforts compared to TD3, But TD3 showed robustness across default hyperparameters in the original setup.
- **Constraint Handling:** All AM-based implementations respected the tire-road friction constraints consistently, demonstrating the strength and flexibility of the Action Mapping mechanism across algorithms.

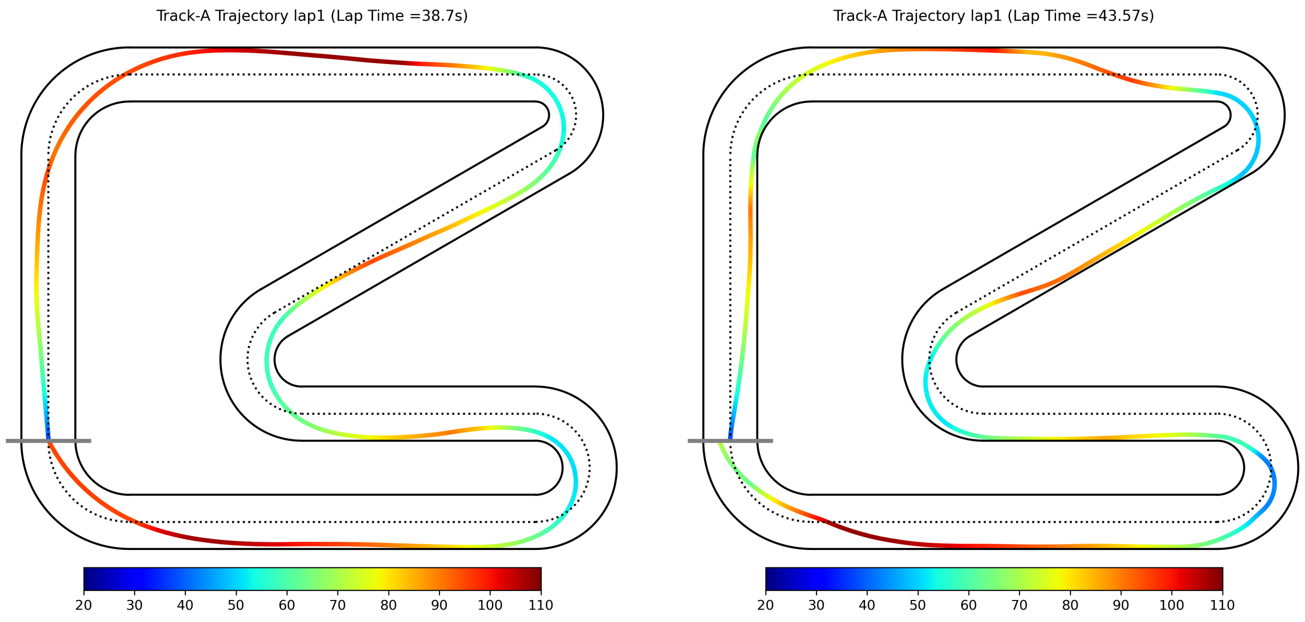


Figure 3.7: Lap 1 performance comparison for TD3 (left) and DDPG (right)

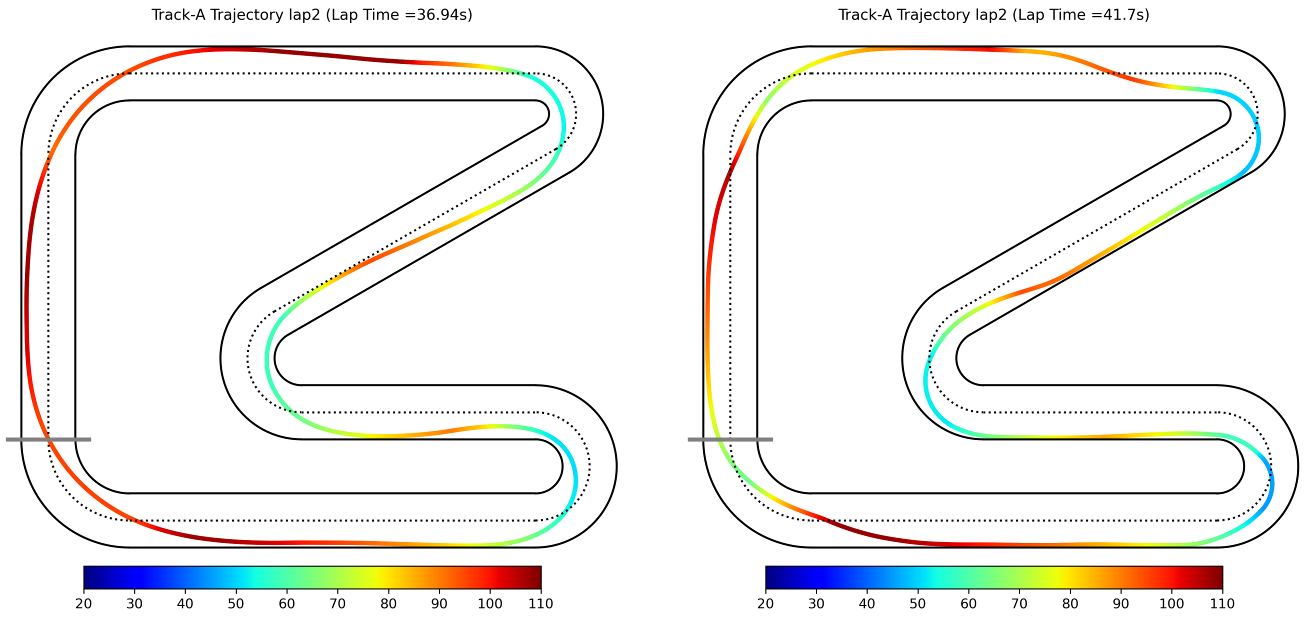


Figure 3.8: Lap 2 performance comparison for TD3 (left) and DDPG (right)

A direct visual and performance-based comparison of the trajectories generated by the DDPG and TD3 algorithms clearly reveals that TD3 produces smoother and more centrally-aligned driving paths. As shown in Figure 3.7 and 3.8, the trajectory learned by TD3 adheres more closely to the center of the track, indicating better control and policy stability. In contrast, the DDPG agent's path tends to oscillate closer to the edges of the track, making it more susceptible to instability or drift, particularly in tight corners. Additionally, TD3 achieves a significantly faster lap time — completing the circuit in 38.7 seconds, whereas DDPG takes 43.57 seconds for the same lap. It is also important to note that TD3 reaches this level of performance using only half the number of training episodes required by DDPG, underscoring its sample efficiency.

These findings strongly reaffirm the theoretical and empirical advantages of TD3 over standard DDPG. TD3 addresses critical weaknesses in DDPG — such as overestimation bias and high variance in policy updates — by incorporating twin critic networks, delayed policy updates, and target policy smoothing. The use of dual critics helps in reducing the Q-value overestimation, while delayed actor updates stabilize learning by giving the critics more time to converge. Target smoothing further regularizes the value function, leading to more consistent and robust policy learning. The performance improvements observed in both trajectory behavior and training efficiency clearly validate the effectiveness of these enhancements, making TD3 a more reliable and performant choice for continuous control tasks like autonomous driving.

# Chapter 4

# Conclusions and Future Work

## 4.1 Conclusion

In this project, we explored the problem of autonomous race driving under strict physical constraints such as limited tire-road friction. Building on the framework proposed in recent literature, we implemented the Action Mapping (AM) mechanism with multiple off-policy actor-critic reinforcement learning algorithms, namely Deep Deterministic Policy Gradient (DDPG) and Soft Actor-Critic (SAC). Our goal was to evaluate whether AM could effectively enforce state-dependent constraints across different learning paradigms and to compare the performance of these algorithms in a realistic racing environment, and compare the results with the original existing implementation with Twin Delayed DDPG (TD3).

We designed and implemented a custom race track simulation using Python and Matplotlib, incorporating a bicycle model of vehicle dynamics and a friction-aware reward function. Our experiments demonstrated that the AM mechanism successfully enforced tire friction constraints, even under high-speed and sharp-turn conditions. The DDPG-based policy learned stable and efficient driving behavior. However, a quick comparison between the implemented DDPG algorithm and the original TD3 algorithm, our algorithm requires twice as much of training time to acheive acceptable results while still underperforming when compared to TD3. While SAC exhibited slower convergence and more sensitivity to reward shaping. However, SAC does not provide us with satisfactory results as it tends to either skid out of the track limits at higher speeds or stop as the acceleration reduces in attempts to stay inside the track, despite both of them yielding us the same high reward value.

Attempts at reshaping the reward function of the system were also made but the resulting model kept the agent constantly stuck at a sub-optimal path on repeated training.

Overall, our work validates the flexibility of Action Mapping as a plug-and-play constraint enforcement mechanism in RL and highlights its applicability across different actor-critic algorithms for safety-critical control tasks like autonomous racing, but still keeps the scope for improved implementation of other sophisticated RL algorithms open.

## 4.2 Future Scope

While our experiments confirms the effectiveness of Action Mapping in constrained RL scenarios, several directions remain open for future exploration:

- **Improving Reward Space for better implementation of SAC**: The Current implementation of SAC does not provide us with satisfactory results. Hence, improving upon the existing reward function could help us in achieving the same.

- **Benchmarking with More Algorithms**: Extending the study to include other RL algorithms, such as A3C, or newer model-based approaches, could offer deeper insights into the generalizability of AM.
- **Sim-to-Real Transfer**: Applying the learned policies to real-world or higher-fidelity simulators (e.g., CARLA, TORCS) would test the robustness of the AM-RL pipeline under real-world uncertainty and noise.
- **Dynamic Friction Modeling**: Incorporating time-varying friction coefficients (e.g., wet or uneven tracks) could make the learning problem more realistic and test the adaptability of AM in more complex scenarios.
- **Multi-agent Race Scenarios**: Implementing AM in multi-agent race settings with competitive or cooperative agents could open avenues for studying emergent behaviors and strategic control under constraints.

By extending in these directions, future work can further establish the scalability, adaptability, and real-world applicability of constraint-aware reinforcement learning strategies in autonomous driving.

# Bibliography

[1] Y. Wang, X. Yuan, and C. Sun, "Learning autonomous race driving with action mapping reinforcement learning," *ISA Transactions*, vol. 150, pp. 1–14, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0019057824002143

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2016. [Online]. Available: https://arxiv.org/abs/1509.02971

[3] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning (ICML)*, 2018. [Online]. Available: http://arxiv.org/abs/1802.09477

[4] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018. [Online]. Available: https://arxiv.org/abs/1801.01290