STAT 4830

Report #1

- Yuv Malik | Mathew Lobo | Pablo Echevarria Cuesta

Project Description:

What Is the Game?

Our project is based on an autonomous racing game in which a simulated racecar must navigate a closed racetrack as efficiently and safely as possible. The agent controls a single vehicle with the goal of completing laps quickly while avoiding collisions with track boundaries. The environment provides partial observations through sensors (e.g., LiDAR), imitating the constraints faced by real autonomous systems that must act based on imperfect, local information rather than full global state.

The task is a sequential decision-making problem under uncertainty, where each action influences future states, rewards, and long-term performance. This makes it a natural setting for reinforcement learning and optimal control techniques.

Why Is This Game Relevant?

Autonomous racing offers us a problem in reinforcement learning and robotics because it combines several challenging elements into a single environment:

- **Continuous state and action spaces**, requiring function approximation.
- **Highly coupled dynamics**, where small control changes can have large downstream effects.
- **Sparse and delayed rewards**, since progress is often measured over time rather than per action.
- **Safety constraints**, such as collision avoidance, that must be balanced against performance objectives like speed.

This problem is closely related to real-world applications, including autonomous driving, mobile robotics, and motion planning. Techniques developed in this simplified racing setting can transfer to broader domains.

Simplifications and Initial Modeling Assumptions

To make the problem tractable within the scope of the course, we intentionally begin with a highly simplified version of the task.

First, the environment is **two-dimensional**. While the simulator uses a physics engine, vertical dynamics (e.g., suspension effects, elevation changes) are ignored, and the car is constrained to planar motion on a flat track.

Second, we **restrict the observation space** by focusing on low-dimensional sensor inputs such as LiDAR scans and basic kinematic information, and we ignore other inputs like raw camera images. This reduces computational complexity and allows us to focus on control and optimization rather than perception.

Third, we consider a **single-agent setting** and ignore interactions with other vehicles. This removes strategic multi-agent effects and allows us to isolate the core control problem before introducing additional complexity later.

Finally, we use a **simplified reward structure**, primarily based on progress along the track and collision penalties, rather than modeling more nuanced objectives such as racing lines or energy efficiency.

These simplifications allow us to validate our reinforcement learning pipeline, understand the learning dynamics, and establish baseline performance. Once this foundation is stable, we can progressively relax assumptions by increasing observation complexity, refining rewards, or extending to multi-agent scenarios in later stages of the project.

RL/Repo Description:

Reinforcement Learning Environment

For our project, we use RacecarGym as the base reinforcement learning environment. RacecarGym is a miniature racecar simulator built on top of the Bullet physics engine (via PyBullet) and is designed to work with standard reinforcement learning interfaces, including the Gymnasium (Gym) API for single-agent settings and the PettingZoo API for multi-agent settings. Instead of building a racing simulator from scratch, we use RacecarGym as the environment our learning agent interacts with, while our project focuses on training and evaluating a new policy within this environment.

RacecarGym simulates an F1Tenth-style racecar driving on a variety of racetracks represented as grid-based occupancy maps. These maps define which areas are drivable and where the walls are, allowing for realistic collision detection and LiDAR (Light Detection and Ranging)-based sensing. Track assets are downloaded automatically the first time they are used, which makes it easy to run experiments on different tracks without manually handling large asset files.

Environment Structure and Design

RacecarGym is structured to clearly separate the simulation logic from the reinforcement learning interface, which makes the environment modular (i.e., flexible and easy to swap or modify components).

Simulation and Physics

The physical behavior of the racecar is handled by a simulation backend that supports Bullet-based physics. Vehicle dynamics are defined using a URDF (Unified Robot Description Format) model, along with mesh geometry and configuration files that specify control limits such as steering range and speed constraints. Together, these components determine how control inputs translate into motion, collisions, and interactions with the track.

Scenarios

Each experiment is defined using a scenario YAML file (a human-readable configuration file format). These files specify the racetrack, the number of agents (cars), the vehicle model used, the sensors available to each agent, and the task used to evaluate performance. This setup allows the same learning approach to be tested across different tracks or configurations without changing the underlying simulator.

Observations

Observations are generated by configurable sensors and returned as a dictionary. Depending on the scenario, an agent may receive information such as pose (position and orientation), velocity, acceleration, LiDAR scans, and optionally RGB camera images. This modular (easily

swappable) observation setup lets us control how much information the agent receives while keeping the rest of the environment unchanged.

Actions

The action space is continuous and normalized to the interval [-1, 1]. By default, the agent controls throttle (motor) and steering, although there is also an option to control target speed and steering instead. These actions are applied through actuators that update the vehicle state within the physics engine at each timestep.

Rewards and Episode Termination

Rewards and termination conditions are defined by task modules. A common objective is to maximize progress along the track while respecting lap limits or time limits. In addition to observations, the environment returns ground-truth state information in the info dictionary (such as progress, collisions, lap number, and rank), which is useful for evaluation and analysis even if it is not directly used by the learning agent.

Role of RacecarGym in Our Project

RacecarGym itself does not train or optimize a policy. Instead, it defines the environment, including the state space, action space, dynamics, reward signals, and episode structure. Our project builds on top of this environment by introducing a learning agent that starts with no prior racing experience and gradually improves its behavior through repeated interaction with the simulator. In this sense, RacecarGym provides the “world,” while our contribution lies in the learning process that takes place within it.

Considerations and Potential Improvements

While RacecarGym provides a strong and flexible foundation, parts of the repository are several years old. This opens up opportunities for improvement, such as updating dependencies to better align with newer Gymnasium tooling, refining reward functions, adjusting sensor or action representations, and exploring more complex multi-agent interactions. Because the environment is modular (designed so components can be changed independently), these improvements can be explored gradually without needing to redesign the entire system.