National University of Computer & Emerging Sciences
Islamabad Campus

Artificial Intelligence
Project Report
Section: D

Group Members:

22i-0891 Umer Farooq
22i-0900 Muhammad Usman
22i-0911 Muhammad Irtaza Khan

1. Introduction

Autonomous car racing provides a compelling domain to test and evaluate AI techniques under real-time constraints. In this project, we developed a neural network-based racing controller for the TORCS (The Open Racing Car Simulator) environment using the Python client provided in the SCR 2015 framework. The objective was to create an intelligent driver capable of completing laps competitively while handling varying racing conditions across different tracks.

TORCS offers a detailed simulation environment with telemetry inputs mimicking real-world sensors. Unlike rule-based systems, which are static and hand-crafted, we designed a data-driven approach using supervised learning, where a neural network learns optimal driving behavior from pre-recorded data.

---

2. Approach

Neural Network Architecture

We used a Recurrent Neural Network (RNN) to model the temporal dependencies in driving decisions. Since driving is inherently sequential and depends on past sensor inputs (e.g., speed, position), RNNs were well-suited for learning these patterns effectively.

Input Data

The input to the model included 29 telemetry parameters:

- **Car state**: Acceleration, Brake, Gear, Steer, Clutch, Focus, Meta, RPM, Fuel, Speed X/Y/Z, Damage, Angle
- **Track & environment**: Track position, Track distance, Opponent proximity, Race position, Distance from start, Distance raced
- **Game stats**: Current lap time, Last lap time, Focus2, Gear2, Wheel spin velocity, Z-axis acceleration

These inputs were captured while manually driving the car in TORCS, which served as our training dataset.

Output Actions

The model outputs five control commands:

- **Steering**
- **Acceleration**
- **Braking**
- **Gear selection**
- **Focus direction**

---

3. Methodology

We adopted a **supervised learning** approach. The training dataset was created by logging telemetry and control data while manually driving the car on various tracks. Each frame in the dataset corresponds to a specific game state and the human driver's response.

The RNN was trained using:

- **TensorFlow** and **PyTorch** frameworks
- **CUDA Toolkit** to leverage GPU acceleration
- **100 epochs**, taking approximately **30 minutes**

Loss functions and optimizers (e.g., Adam) were used to minimize the error between predicted and actual actions.

---

4. Results & Evaluation

While the TORCS race environment includes collision and off-track detection, we focused our performance metrics on:

- **Lap time**
- **Consistency across tracks**
- **Smoothness of control transitions**

The model was successfully able to:

- Follow track curvature
- Handle variable speed control
- Shift gears with learned logic (after correcting an early bug)

Despite minor inconsistencies, especially on sharp turns or during crowding by opponents, the controller showed promise and demonstrated autonomous lap completion.

---

5. Challenges and Future Work

- **Gear Prediction Bug**: Initially, the model always predicted gear "5," resulting in zero performance. This was fixed by adjusting gear encoding in the training data.
- **Data Loss**: During logging, intermittent sensor recording failures led to some missing data. Future iterations should include better validation and real-time sanity checks.
- **Model Generalization**: The current model overfits to tracks seen during training. We aim to augment the dataset with diverse tracks and include LSTM cells to enhance temporal learning.

Future Improvements

- Use of **Reinforcement Learning** (e.g., DDPG or PPO) for long-term reward optimization
- Implementing **attention mechanisms** to focus on critical sensors
- Real-time adaptive controllers using hybrid models (RNN + rule-based safety layer)

---

6. Conclusion

This project successfully demonstrated how a neural network, specifically an RNN, can learn to drive a car in a simulated environment using supervised learning. With continued improvements and extended training data, such controllers can approach or even surpass rule-based systems in performance and adaptability.