FAST NUCES, Islamabad

ARTIFICIAL INTELLIGENCE

Final Project Report

Group Members:

Afrah Syed	**| i22-1008**
Abdur Raheem	**| i22-0777**
Areeba Riaz	**| i22-1244**

Section: **BSCS-H**

Autonomous Car Racing Controller using Feedforward Neural Network (FFNN) in TORCS

Introduction

In this project, we worked with **TORCS** (The Open Racing Car Simulator), a realistic car racing simulator that is often used for AI training and research. One of the cool things about TORCS is how it's designed. It uses a **client-server architecture**, meaning the game runs as a server, and bots (drivers) connect as clients through **UDP connections**. Every 20 milliseconds of simulated time, the server sends the bot sensory input — like the car's speed, position, and the distance to track edges — and waits up to 10 milliseconds in real time for the bot to respond with an action. If the bot doesn't reply in time, TORCS just keeps using the last action the bot took.

Another important feature is the **abstraction layer** that separates the racing simulation from the bot's code. This gives flexibility in choosing the programming language for the bots — sample clients are provided in Python, Java, C++, and C#. For this project, we used the **Python client**, as it was the most convenient for integrating machine learning models.

Project Goal

Our main goal was to design a car racing controller that could perform well in a competitive environment — meaning it should be fast, avoid obstacles, stay on track, and handle different types of race circuits. We aimed to build this controller using a **Feedforward Neural Network (FFNN)**, a basic type of neural network where information moves in one direction — from input to output — without looping back.

Approach and Implementation

We started by understanding the telemetry data sent from the TORCS server. This includes sensor readings like:

- Car's speed and angle
- Distance from the track edges (track sensors)

- Opponent proximity (opponent sensors)
- Current gear, RPM, and fuel level

We used this sensory input as features for our FFNN. The network's job was to learn how to output driving actions like **acceleration, braking, and steering angle** based on this input. For training data, we initially ran some laps using rule-based driving to collect sensor-action pairs. Then, we used this data to train the FFNN.

The architecture of our FFNN was simple — just a few fully connected layers with activation functions like ReLU. The final output layer used a sigmoid or tanh depending on whether the action values needed to be in a specific range (e.g., -1 to 1 for steering).

We also normalized all input features for better training stability and used mean squared error as our loss function.

Results and Observations

Once trained, our controller was able to complete laps without crashing or going off-track. It responded well to curves and straights, adjusting speed accordingly. While it wasn't perfect — and could sometimes oversteer or understeer — it showed promising behavior, especially on tracks it had seen during training.

One key observation was that FFNNs don't have memory, so the car didn't adapt well to new or complex track patterns. In the future, using a model with memory, like an RNN or LSTM, could help with learning long-term strategies.

Conclusion

Overall, this project gave us hands-on experience with both machine learning and real-time simulation. Using FFNNs for autonomous driving in TORCS helped us understand how sensor data can be mapped to control actions. We successfully built a controller that could navigate a race track and perform basic driving tasks. With more training data, tuning, and advanced models, the controller could become even more competitive.