Artificial Intelligence
TORCS Hybrid Driving Model

Ayaan Khan (22i-0832)
Ayaan Mughal (22i-0861)
Malaika Afzal (22i-0885)

Artificial Intelligence Course Project
FAST NUCES Islamabad

May 11, 2025

Project Repository

GitHub: `https://github.com/AyaanKhan1576/TORCS-Python-Client`

1 Overview

This project integrates manual keyboard driving and machine learning-based control in the open-source TORCS racing simulator. The system enables users to drive vehicles using the keyboard while simultaneously logging sensor and control data, which is then used to train a machine learning model. Once trained, the model can take over control of the vehicle in real-time, mimicking human-like driving behavior.

2 Project Components

- **pyclient.py**: The main client script that connects to the TORCS server, manages episodes, and interacts with the `Driver` class.
- **driver.py**: Contains the core logic for controlling the car via keyboard or machine learning, sensor parsing, gear control, and CSV data logging.
- **train_torch.py**: A PyTorch training script that loads CSV driving data and trains a neural network on recorded sensor-action pairs.
- **models.py**: Defines the machine learning model architecture used for steering, throttle, and brake prediction.

- **utils.py**: Utility functions for preprocessing data, standardizing input values, and scaling features.

3 Execution Instructions

To launch the driver and begin a TORCS episode:

```
python pyclient.py --port 3001 --stage 0 --maxEpisodes 1 --maxSteps 0
```

This starts a single driving session using either the keyboard or a pre-trained model.

4 Machine Learning Model Details

The core of the AI control in this project is a supervised learning model built using PyTorch. It is a feedforward neural network that takes as input a vector of real-valued driving sensors (speed, angle, RPM, track sensors, etc.) and categorical embeddings (track and car names), and outputs three real-valued control signals: steering, acceleration, and brake.

The network consists of:

- Input: 93 numerical sensor inputs
- Embeddings: For track and car names
- Fully connected layers with ReLU activations
- Output layer with 3 values (steer, accel, brake)

We trained this model on data logged while manually driving the car. Each row in the dataset corresponds to one tick of simulation and includes the car's sensor values and the control keys pressed. The training process used mean squared error loss and the Adam optimizer.

Why this model?

We chose a supervised feedforward neural network over more complex models like reinforcement learning or recurrent architectures for the following reasons:

- Simplicity: It is easier to debug and train.
- Data availability: Supervised learning works directly with our labeled dataset of sensor-action pairs.
- Real-time inference: Feedforward networks are fast and suitable for real-time TORCS control.
- Proven effectiveness: Previous research has shown that basic behavior cloning works well in TORCS for short-term control.

5 Dataset Format

Each row in the training CSV contains:

- Sensor values: 93 columns (speedX, angle, trackPos, track0–track18, focus0–4, etc.)
- Control outputs: steering, accel, brake
- Meta-data: clutch, gear, reset state, focus direction, keypresses

This dataset is parsed, scaled using min-max normalization, and split into sequences of length 1 (stateless prediction).

6 Challenges Faced

- **Data Quality:** Initially, some sensor data such as focus and opponents were logging NaNs. This was fixed by checking sensor defaults and replacing missing values.
- **Gear logic bugs:** Gear shifting had to be decoupled from model outputs to preserve realistic acceleration.
- **Model mismatch:** Some training models used different architectures. We ensured consistency between training and inference-time models.
- **Oversteering:** We had to tune the model and input features to avoid unstable behavior in turns.

7 Conclusion

This project successfully combines manual and AI-based driving in TORCS using a Python client and a neural network. It demonstrates core concepts of behavior cloning, supervised learning, sensor-action mapping, and real-time inference in a simulated environment.