TORCS AI Project Report: Developing an Autonomous Driving System

Rumman Qadir (22i-1204), Awais Ahmed (22i-1225), Ahmed Asif (22i-1299)

May 11, 2025

1 Introduction

This report chronicles our development of an autonomous driving system for the TORCS (The Open Racing Car Simulator) competition, using imitation learning with a multi-headed MLP neural network. Starting with a manual driver, we iteratively refined our approach through data collection, model design, and preprocessing, overcoming challenges like gear oscillation and constant braking. We detail our journey, technical implementation, and team contributions.

2 Project Development Journey

2.1 Initial Manual Driver

We began by implementing a manual driver in `driver.py`, controlling the SCR bot with hand-crafted rules for gear changes, acceleration, braking, and steering. This allowed us to understand TORCS's telemetry data (e.g., angle, speedX, rpm, trackPos) and action commands (accel, brake, steer, gear). However, manual gear shifts were inconsistent, prompting us to refine the control logic for smoother driving.

2.2 Improved Manual Control and Data Collection

We enhanced the manual driver to achieve better control, adjusting rules for more precise steering and acceleration based on track position and speed. Using this improved driver, we collected an initial small dataset, logging sensor and action data into `client_log.csv`. This dataset, though limited, served as our starting point for machine learning experiments.

2.3 Early Machine Learning Attempts

Our first ML model was a simple neural network with four output neurons (accel, brake, steer, gear), implemented in PyTorch. Trained on the small dataset,

this model struggled with gear predictions, producing erratic shifts due to treating gear as a continuous output. Recognizing the need for separate handling of continuous and discrete outputs, we transitioned to a multi-headed MLP architecture.

2.4 Multi-Headed MLP Development

We designed a new model (`DrivingMLP`) with a shared backbone and two heads:

- `control_head`: Outputs continuous values for accel, brake, and steer.
- `gear_head`: Outputs 8 logits for gear classification (-1 to 6, mapped to 0 to 7).

Initially, the backbone had two hidden layers (128 neurons each, ReLU activations). Training with `train.py` (Adam optimizer, 200 epochs, MSE loss for controls, Cross-Entropy loss for gears) showed promise but lacked robustness. We increased to three hidden layers to capture more complex patterns, tested 256 neurons per layer, but reverted to 128 for better convergence and computational efficiency.

2.5 Data Preprocessing and Scaling

To improve performance, we introduced preprocessing in `data_utils.py`. We filtered out pre-race data (negative `curLapTime`) and normalized sensor features to [0, 1] using `MinMaxScaler`, saving the scaler as `scaler.pkl` for inference in `pyclientai.py`. Despite these changes, initial results were poor, with issues like gear oscillation and constant braking (e.g., brake values 0.12), likely due to limited data diversity.

2.6 Scaling Up Data and Addressing Imbalance

We collected a larger dataset with the refined manual driver, capturing more varied driving scenarios. Analysis revealed a brake imbalance (97% `brake = 0.0`, 3% non-zero), causing the model to predict small positive brake values, slowing the car. To address this, we:

- Added thresholding in `pyclientai.py` to snap brake values <0.15 to 0.0, eliminating unintended braking.
- Adjusted loss weights in `train.py` to emphasize non-zero brake predictions, balancing control outputs.

These changes, combined with training on the larger dataset, enabled the car to move effectively, with stable gear shifts and improved acceleration.

2.7 Final System

The final system uses the multi-headed MLP trained on the large, preprocessed dataset. Inference in `pyclientai.py` applies normalized features and thresholded brake outputs, ensuring smooth driving. Testing showed significant improvements, with the car navigating tracks without constant braking or gear oscillation, positioning us for strong competition performance.

3 Technical Implementation

3.1 Telemetry Processing

The `driver.py` script logs 19 sensor fields (e.g., angle, rpm, speedX) and 4 action fields (accel, brake, steer, gear) into `client_log.csv`. `data_utils.py` preprocesses this data by filtering invalid rows, normalizing features, and correctly loading action gears, ensuring high-quality inputs for training.

3.2 Model Architecture

The `DrivingMLP` in `model.py` features a three-layer backbone (128 neurons, ReLU) and two heads: one for continuous controls (3 outputs) and one for gear classification (8 outputs). Training in `train.py` uses a split dataset (80% training, 20% validation), with balanced MSE and Cross-Entropy losses.

3.3 Inference Pipeline

The `pyclientai.py` script connects to TORCS via UDP, processes sensor messages, normalizes features with `scaler.pkl`, and predicts actions. Brake thresholding and gear mapping (-1 to 6) ensure valid commands, achieving robust real-time performance.

4 Team Contributions

All members contributed equally:

- Rumman Qadir (22i-1204): Developed the manual driver, data collection, and preprocessing in `data_utils.py`.
- Awais Ahmed (22i-1225): Designed the multi-headed MLP and implemented training in `train.py`.
- Ahmed Asif (22i-1299): Built the inference pipeline in `pyclientai.py` and debugged issues like brake imbalance.

5 Conclusion

Our TORCS AI evolved from a manual driver to a robust imitation learning system, overcoming challenges through iterative model design, data scaling, and preprocessing. Key achievements include transitioning to a multi-headed MLP, implementing normalization, and resolving brake imbalance. The final system drives effectively, ready for competition. Future improvements could involve collecting more diverse braking data to enhance prediction accuracy.