**TORCS AI PROJECT**

**Team:**

**Iqrash Qureshi (22i1174)**

**Daniyal Khawar (22i1066)**

**Hammad Shabbir (22i1140)**

## Executive Summary

This project implements an AI-driven racing agent for The Open Racing Car Simulator (TORCS) using machine learning techniques. The system features a neural network model trained on collected telemetry data to predict optimal driving controls. The implementation includes both autonomous AI operation and human control interfaces (keyboard and game controller), with comprehensive data logging capabilities for further model refinement.

## Table of Contents

1. [Project Overview](#project-overview)

2. [System Architecture](#system-architecture)

3. [Data Collection Methodology](#data-collection-methodology)

4. [Neural Network Approach](#neural-network-approach)

5. [Training Process](#training-process)

6. [Control Interfaces](#control-interfaces)

7. [Results and Performance](#results-and-performance)

8. [Limitations and Future Work](#limitations-and-future-work)

## Project Overview

The project aims to develop an autonomous racing agent for TORCS that can effectively navigate tracks and compete in races. The system implements two main control modes:

1. \*AI Control\*: A neural network model predicts optimal acceleration, braking, steering, and gear shifting based on sensor data.

2. \*Human Control\*: Interfaces for keyboard and game controller input allowing human drivers to control the vehicle while collecting telemetry data.

A comprehensive logging system captures all sensor data and control inputs, enabling continuous improvement of the AI model through supervised learning.

## System Architecture

The system consists of three main components:

### 1. Driver Interface (driver.py)

The primary interface between TORCS and the control logic with the following features:

- Initializes communication with the TORCS simulator

- Implements multiple control modes (AI, keyboard, controller)

- Collects and logs comprehensive sensor data

- Handles user input processing from keyboard/controller

- Implements baseline AI driving logic for comparison

### 2. Data Processing Pipeline

Processes the raw telemetry data:

- Collects sensor readings and corresponding control actions

- Organizes data into structured CSV format

- Provides utility scripts for data validation and column extraction (get\_column\_names.py)

### 3. Neural Network Training System (training.py)

Implements machine learning pipeline:

- Data loading and preprocessing

- Feature selection and scaling

- Neural network model definition

- Training and evaluation processes

- Model serialization for deployment

- Custom NNDriver class for deploying trained models

## Data Collection Methodology

The system employs a comprehensive data collection approach to capture all relevant aspects of the racing environment and driver responses:

### Sensor Data Collected

- \*Track Information\*: 19 rangefinder sensors measuring distance to track edges

- \*Car State\*: Speed (X, Y, Z components), angle, track position, rpm, gear

- \*Race Information\*: Lap times, race position, distance raced, damage

- \*Additional Sensors\*: Opponent positions, focus sensors, wheel spin velocity

### Control Data Captured

- Acceleration input (0.0-1.0)

- Braking input (0.0-1.0)

- Steering input (-1.0 to 1.0)

- Gear selection (1-6)

- Raw input events (keyboard keys, controller buttons/axes)

### Logging Implementation

The logging system writes data to time-stamped CSV files with:

- Full sensor state for each frame

- Control outputs generated (by AI or human)

- Input events from human controllers

- Timestamps for synchronization

## Neural Network Approach

The project utilizes a supervised learning approach with a Multi-Layer Perceptron (MLP) neural network to predict optimal control values from sensor inputs:

### Model Architecture

- \*Type\*: MLPRegressor from scikit-learn

- \*Hidden Layers\*: Three layers with 256, 128, and 64 neurons respectively

- \*Activation Function\*: ReLU (Rectified Linear Unit)

- \*Solver\*: Adam optimizer

- \*Early Stopping\*: Enabled with validation fraction of 0.1

- \*Batch Size\*: 32 samples

### Feature Selection

The model uses the following features as inputs:

- Track sensor readings (track\_0 through track\_18)

- Track position (trackPos)

- Car angle relative to track

- Speed components (speedX, speedY, speedZ)

- Engine RPM

- Current gear

### Target Variables

The model is trained to predict four continuous control values:

1. Acceleration (accel)

2. Braking (brake)

3. Steering (steer)

4. Gear selection (gear)

## Training Process

The training pipeline follows standard machine learning practices with specific adaptations for the racing domain:

### Data Preprocessing

1. Feature extraction from raw CSV logs

2. Train-test split (80% training, 20% testing)

3. Feature standardization using sklearn's StandardScaler

4. Serialization of scaling parameters for deployment

### Model Training

- Hyperparameters tuned for the racing domain

- Adaptive learning rate for better convergence

- Early stopping to prevent overfitting

- Maximum of 1000 iterations

- L2 regularization (alpha=0.0001)

### Evaluation Metrics

Performance is evaluated using:

- Mean Squared Error (MSE) for each target variable

- R² Score to measure explained variance

- Separate metrics reported for each control output

## Control Interfaces

The system implements three control modes to facilitate both data collection and autonomous operation:

### 1. AI Control Mode

- Original baseline AI: Rule-based approach for basic navigation

- Neural network AI: Trained model predicting optimal controls

- Intelligent gear shifting based on RPM trends

### 2. Keyboard Control Interface

- Arrow keys for steering and acceleration/braking

- Gear shifting with Z (down) and X (up) keys

- Event tracking for comprehensive input logging

### 3. Game Controller Support

- Left stick for analog steering control

- Right trigger for acceleration (proportional)

- Left trigger for braking (proportional)

- A/B buttons for gear shifting

- Input value logging with precision (e.g., "STEER\_0.75")

## Results and Performance

The neural network model demonstrates the following performance characteristics:

### Model Performance

- Performance metrics calculated separately for each control output

- MSE and R² values provide insights into prediction accuracy

- Explicit evaluation on held-out test data

### Driver Implementation

The NNDriver class combines the trained model with TORCS integration:

- Sensor parsing to extract relevant features

- Feature preprocessing using the saved scaler

- Real-time prediction of control outputs

- Formatting outputs for TORCS compatibility

## Limitations and Future Work

While the current implementation provides a functional framework, several improvements could enhance performance:

### Current Limitations

- Fixed neural network architecture may not capture all racing dynamics

- Limited feature engineering beyond raw sensor values

- Single model predicting multiple outputs instead of specialized models

- Basic handling of gear shifting

### Future Enhancements

- Implementing more advanced neural network architectures (CNN, LSTM)

- Exploring reinforcement learning approaches

- Creating separate models for different control aspects

- Adding track-specific optimizations

- Implementing competitive racing strategies (overtaking, defending)

- Expanding the dataset with more diverse driving scenarios

- Real-time visualization of AI decision-making process

## Conclusion

This project successfully demonstrates the application of supervised machine learning to autonomous racing in TORCS. The comprehensive data collection system and flexible control interfaces provide a solid foundation for further research and development in racing AI. The neural network approach shows promise in capturing the complex relationships between sensor inputs and optimal control outputs, with room for continued refinement through additional training data and model improvements.