Graphical user interface

Description automatically generated with medium confidence

# **ARTIFICIAL INTELLIGENCE**

# PROJECT FINAL DELIVERABLE (DUE : 11-MAY-2025)

# AN ASSIGNMENT PRESENTED TO:

# ALI ZEESHAN

|  |  |  |
| --- | --- | --- |
| SR. | NAME | ROLL NO. |
| 1. | M. ZAYYAM HASSAN | 22I-1247 |
| 2. | M. RAYYAN | 22I-1022 |
| 3. | UMER FAROOQ | 22I-1007 |

# GROUP MEMEBERS

**TORCS Racing Game AI Controller: Technical Documentation**

**1. Introduction**

The TORCS (The Open Racing Car Simulator) platform offers a sophisticated environment for developing autonomous racing controllers. TORCS functions as a client-server application where racing bots connect to the race server via UDP connections. The system operates in real-time, with the server sending sensory inputs to each bot every game tic (approximately 20ms of simulated time) and waiting 10ms for a response before continuing the simulation.

Our project aimed to develop an advanced AI controller capable of racing effectively against other cars across various tracks. Using deep learning techniques, specifically LSTM neural networks, we created a controller that processes telemetry data to make real-time decisions about steering, acceleration, and braking.

**2. Project Objectives**

The primary objectives of this project were to:

* Design an intelligent controller for autonomous car racing in TORCS
* Implement effective track following behavior
* Develop obstacle avoidance capabilities
* Optimize racing performance through machine learning
* Create a model that can generalize to different tracks and racing conditions

Performance metrics focused on racing speed, adherence to track limits, and safe navigation around competitors.

**3. Technical Implementation**

**3.1 Data Processing Pipeline**

Our implementation leverages a sophisticated data processing pipeline to transform raw sensor data into meaningful features for our neural network model. The pipeline consists of several key stages:

**3.1.1 Data Loading and Initial Preprocessing**

The system begins by loading racing data from the "Dataset.csv" file, which contains recorded telemetry from expert racing sessions. This dataset includes raw sensor readings such as:

* **Basic Car Metrics**: RPM, SpeedX/Y/Z, TrackPosition, Z position
* **Control Inputs**: Steering, Acceleration, Braking values
* **Track Sensors**: 19 distance sensors (Track\_1 through Track\_19) that detect track boundaries

We clean column names by stripping whitespace and select the relevant features for our model.

**3.1.2 Feature Engineering**

To enhance the model's understanding of the racing environment, we engineered additional features:

* **Speed**: Calculated as the Euclidean norm of the three-dimensional speed vector (SpeedX, SpeedY, SpeedZ)
* **TrackWidth**: Sum of Track\_1 and Track\_2 sensors, providing information about the available space
* **UpcomingCurvature**: Average of Track\_3, Track\_4, and Track\_5 readings, offering insight into approaching turns

These engineered features provide higher-level abstractions that help the model understand the racing environment more effectively.

**3.1.3 Normalization**

To ensure stable and efficient training, we apply robust scaling to key features:

* RPM
* SpeedX
* Speed (calculated)
* TrackWidth (calculated)
* UpcomingCurvature (calculated)

We use RobustScaler from scikit-learn, which scales features using statistics that are robust to outliers, improving the model's ability to handle varying track conditions and driving scenarios.

**3.1.4 Sequence Creation**

Since racing involves temporal dependencies (current decisions affect future states), we structure the data as sequences using a sliding window approach:

* Each input sample consists of SEQ\_LENGTH (5) consecutive time steps of features
* Each output sample consists of the target control values (Steering, Acceleration, Braking) for the next time step

This sequence-based approach allows the model to learn patterns over time, making it more effective at anticipating turns and reacting to changing conditions.

**3.2 Neural Network Architecture**

Our racing controller is powered by a sophisticated deep learning model built with TensorFlow. The architecture incorporates:

**3.2.1 Long Short-Term Memory (LSTM) Layers**

The core of our model consists of stacked LSTM layers:

* An initial LSTM layer with 128 units that maintains sequence information
* A second LSTM layer with 64 units that further processes the temporal patterns

LSTM networks are particularly suited for this application as they can:

* Capture long-term dependencies in sequential data
* Remember relevant information across multiple time steps
* Filter out noise while preserving important patterns in driving behavior

**3.2.2 Dense Layers**

Following the LSTM layers, we implement several dense (fully connected) layers:

* A 64-neuron dense layer with ReLU activation
* A dropout layer (rate=0.2) to prevent overfitting
* A 32-neuron dense layer with ReLU activation
* A final output layer with 3 neurons (one for each control: steering, acceleration, braking)

The output layer uses linear activation to allow the network to produce values across the full range needed for the control signals.

**3.2.3 Model Training**

The training process incorporates several optimization strategies:

* Adam optimizer with a learning rate of 0.001
* Mean Squared Error (MSE) loss function
* Mean Absolute Error (MAE) as a monitoring metric
* Early stopping to prevent overfitting (patience=10 epochs)
* Learning rate reduction on plateau (factor=0.5, patience=5)
* Batch size of 64 samples
* Maximum of 100 epochs (typically terminated earlier by early stopping)

**3.3 Model Optimization and Deployment**

To ensure our model runs efficiently in the real-time environment of TORCS, we implement several optimization techniques:

**3.3.1 TensorFlow Lite Conversion**

We convert the trained model to TensorFlow Lite format, which:

* Reduces model size
* Improves inference speed
* Enables deployment on resource-constrained environments

The conversion process includes:

* Float16 quantization for reduced memory footprint
* Optimization for default operations
* Support for TensorFlow Lite built-ins and select TensorFlow operations

**3.3.2 Fallback Mechanisms**

We implemented robust error handling to ensure reliability:

* If TensorFlow Lite conversion fails, the system falls back to saving in the standard SavedModel format
* Comprehensive logging throughout the pipeline provides debugging information

**3.4 Racing Controller Implementation**

The RacingController class forms the interface between our trained model and the TORCS environment:

**3.4.1 State Preprocessing**

For each game tick, the controller:

1. Processes the raw sensor data from TORCS
2. Applies the same feature engineering used during training
3. Normalizes features using the saved scaler
4. Maintains a buffer of recent states to form the required sequence length

**3.4.2 Prediction and Action Selection**

Once sufficient history is available, the controller:

1. Reshapes the buffered data to match the model's input requirements
2. Uses the TensorFlow Lite interpreter to generate predictions
3. Processes the raw predictions into valid control signals:
   * Steering is clipped to the range [-1, 1]
   * Acceleration is clipped to the range [0, 1]
   * Braking is clipped to the range [0, 1]
4. Applies a simple rule: if braking exceeds 0.3, acceleration is set to 0

This ensures the controller produces physically realistic and safe control inputs.

**4. Challenges and Solutions**

Throughout the development of our TORCS racing AI controller, we encountered and overcame several significant challenges:

**4.1 Data Processing Challenges**

**Challenge**: Handling the high-dimensional and noisy sensor data from TORCS efficiently.

**Solution**:

* Created targeted feature engineering to extract meaningful information
* Implemented robust scaling to handle outliers in sensor readings
* Used sequence-based preprocessing to capture temporal relationships

**4.2 Model Architecture Optimization**

**Challenge**: Designing a network architecture that could process sequential data and make real-time decisions.

**Solution**:

* Experimented with various LSTM configurations
* Added layer normalization to improve training stability
* Implemented early stopping and learning rate reduction to prevent overfitting
* Fine-tuned hyperparameters including sequence length, batch size, and layer dimensions

**4.3 Real-time Performance Constraints**

**Challenge**: Ensuring the controller could process inputs and generate outputs within the strict 10ms time window imposed by TORCS.

**Solution**:

* Converted the model to TensorFlow Lite for optimized inference
* Applied float16 quantization to reduce computational overhead
* Implemented buffer management to minimize redundant processing
* Created a fallback mechanism for when TFLite conversion failed

**4.4 Driving Behavior Balancing**

**Challenge**: Finding the right balance between aggressive racing (high speed) and safe driving (staying on track, avoiding collisions).

**Solution**:

* Engineered features specifically targeting track awareness (TrackWidth, UpcomingCurvature)
* Implemented post-processing rules for control outputs (e.g., no acceleration during heavy braking)
* Used multiple sensor readings to create a comprehensive understanding of the racing environment

**5. Team Contributions**

**5.1 Muhammad Zayyam Hassan (Core Implementation, Logic Building, and Model Training)**

* Designed and implemented the neural network architecture
* Created the feature engineering pipeline
* Developed the TensorFlow model training and optimization workflow
* Implemented the RacingController class for real-time decision making
* Conducted hyperparameter tuning experiments
* Integrated the model with the TORCS client interface

**5.2 Muhammad Rayyan (Documentation, Testing, and Model Training)**

* Performed extensive testing of the controller across different tracks
* Analyzed model performance and suggested architectural improvements
* Assisted in training multiple model variants to find optimal configurations
* Created testing protocols to evaluate controller performance
* Documented the technical implementation and system architecture
* Helped optimize the TensorFlow Lite conversion process

**5.3 Umer Farooq (Data Collection)**

* Gathered training data from expert racing sessions
* Created and maintained the Dataset.csv file
* Explored different data collection strategies to improve model performance
* Helped with feature engineering experiments

**6. Results and Evaluation**

Our final controller demonstrates several key capabilities:

* **Effective Track Following**: The controller successfully stays within track boundaries across various circuits
* **Speed Optimization**: The model learns to balance speed and safety, maximizing velocity on straightaways while appropriately slowing for curves
* **Adaptive Control**: Through the LSTM-based architecture, the controller demonstrates anticipatory behavior, reacting to upcoming track features before they become immediate

Performance metrics showed:

* Consistent lap times across multiple testing sessions
* Successful navigation of challenging track sections
* Appropriate use of acceleration, braking, and steering controls

**7. Conclusion and Future Work**

Our implementation successfully demonstrates the effectiveness of using deep learning, specifically LSTM neural networks, for autonomous racing control in TORCS. By combining feature engineering, sequence-based learning, and model optimization, we created a controller capable of real-time decision making in a complex racing environment.

**Future Improvements**

Several areas for future enhancement include:

* **Reinforcement Learning**: Implementing a reinforcement learning approach to further optimize racing lines and racing strategy
* **Opponent Awareness**: Enhancing the model to specifically account for the presence and behavior of competitor vehicles
* **Transfer Learning**: Developing methods to quickly adapt the model to new tracks with minimal additional training
* **Enhanced Feature Engineering**: Creating more sophisticated derived features from the raw sensor data
* **Hardware Acceleration**: Exploring dedicated hardware acceleration to further improve inference speed

The project provides a solid foundation for ongoing research and development in autonomous racing systems, with potential applications beyond gaming environments into real-world autonomous vehicle control.