# TORCS Racing Car Project Report

Ali Usman 22i-0926
Abbas Ali 21i-0773
Mattee ul Haq 22i-0989

## Introduction

The Autonomous Racing Car Project develops a machine learning-based controller for a virtual racing car in the TORCS (The Open Racing Car Simulator) environment. The system leverages polynomial regression models to predict control actions (acceleration, steering, braking, and gear shifting) based on sensor data, enabling the car to navigate a track autonomously. The project integrates a Python client (`pyclient.py`) that communicates with the TORCS server, a driver module (`driver.py`) for heuristic control, and a training script (`regression_training.py`) to build predictive models.

## System Overview

The project consists of several key components:

1. **TORCS Server Communication (`pyclient.py`)**:
   - Establishes a UDP connection with the TORCS server to exchange sensor data and control commands.
   - Receives sensor data (e.g., track position, angle, speed, RPM) and processes it into a feature vector.
   - Uses trained machine learning models to predict control actions: acceleration (`Accel`), steering (`Steer`), braking (`Brake`), and gear (`Gear`).
   - Sends control commands back to the TORCS server to drive the car.
2. **Driver Module (`driver.py`)**:
   - Implements heuristic control logic for steering, gear shifting, and speed management.
   - Provides fallback behavior and initializes the car's sensor angles for track detection.
   - Processes sensor data via `carState.py` and formats control commands via `carControl.py`.
3. **Training Script (`regression_training.py`)**:

   - Trains polynomial regression models using historical driving data stored in a CSV file (`lancer oval2.csv`).
   - Generates four models for acceleration, steering, braking, and gear shifting, saved as `.sav` files for use by `pyclient.py`.
4. **Utility Modules**:
   - `carState.py`: Parses and stores car state variables (e.g., speed, track position, angle).
   - `carControl.py`: Manages control parameters (e.g., acceleration, brake, gear) and formats them into messages.
   - `msgParser.py`: Handles parsing of UDP messages from the TORCS server and stringifying control commands.

## How the Project Works

The system operates in a loop to control the car autonomously:

1. **Initialization**:
   - `pyclient.py` loads four pre-trained models (`acc_controller.sav`, `steering_controller.sav`, `brake_controller.sav`, `gear_controller.sav`) from the `../training/models/` directory.
   - It establishes a UDP socket connection to the TORCS server using configurable parameters (host, port, bot ID, etc.).
   - The `Driver` class in `driver.py` initializes the car's sensor angles and control parameters.
2. **Data Acquisition**:
   - The TORCS server sends sensor data (e.g., track distances, speed, RPM, angle) via UDP.
   - `pyclient.py` parses the data using `carState.py` and extracts 11 features:
     - `S1` (`track_1`): Distance to track edge at sensor 1.
     - `S17` (`track_17`): Distance at sensor 17.
     - `S9` (`track_9`): Distance at sensor 9.
     - `TPos` (`trackPos`): Track position relative to the center.
     - `TAngle` (`angle`): Angle of the car relative to the track.
     - `PGear` (`prev_gear`): Previous gear.
     - `RPM` (`rpm`): Engine RPM.
     - `PSpeed` (`speedX`): Speed in the X direction.
     - `DIff` (`abs_diff_track_angle_pos`): Absolute difference between track position and angle.

        - `PrevBreak` (`prev_brake`): Previous brake value.
        - `CurrBreak` (`brake`): Current brake value.
    - These features are stored in a DataFrame and converted to a NumPy array (`X`, shape `(1, 11)`).
3. **Feature Transformation**:
    - For `accel`, `steer`, and `brake` predictions, `pyclient.py` applies `PolynomialFeatures(degree=2, include_bias=False)` to `X`, transforming the 11 features into 77 polynomial features (including linear, quadratic, and cross terms).
    - The gear model uses the raw 11 features (`X`) directly.
4. **Prediction and Control**:

The transformed features (77 features) are fed to the `accel`, `steer`, and `brake` models to predict control values:

Accel = modelAccel.predict(poly_features)
Brake = modelBrake.predict(poly_features)
Steer = modelSteer.predict(poly_features)

    -

The raw features (11 features) are used for the gear model:

Gear = modelGear.predict(X)

    -
    - Predicted values are rounded (e.g., `Brake` to one decimal place) and set as control parameters using `carControl.py`.
    - The control commands are formatted into a UDP message and sent to the TORCS server.

5. **Loop Continuation**:
    - The loop continues until the maximum episodes or steps are reached, or the server signals shutdown or restart.
    - The car navigates the track autonomously, adjusting its controls based on real-time predictions.

## Training Process

The training process is implemented in `regression_training.py` and uses a dataset (`lancer oval2.csv`) to train polynomial regression models.

## Why Polynomial Regression Was Used

Polynomial regression was chosen for the following reasons:

1. **Nonlinear Relationships**: The relationship between sensor inputs (e.g., track position, speed, angle) and control outputs (e.g., acceleration, steering) is nonlinear. Polynomial features capture these relationships by including quadratic and cross terms.
2. **Simplicity and Interpretability**: Polynomial regression is computationally efficient and easier to interpret compared to complex models like neural networks, making it suitable for real-time control in TORCS.
3. **Feature Expansion**: The 11 input features are expanded to 77 polynomial features, allowing the model to learn complex interactions (e.g., the combined effect of track position and angle on steering).
4. **Gear Prediction Simplicity**: Gear shifting depends primarily on linear relationships with features like RPM and speed, so a simpler model (raw features) suffices.

## How the Training Process Works

1. **Data Loading and Preprocessing**:
    - The CSV file (`lancer oval2.csv`) is loaded into a Pandas DataFrame.
    - Missing or infinite values are removed to ensure data quality.
    - The 11 features are extracted, with derived features computed:
        - `prev_brake`: Brake value from the previous time step (shifted and filled with 0 for the first row).
        - `prev_gear`: Gear from the previous time step.
        - `abs_diff_track_angle_pos`: Absolute difference between `trackPos` and `angle`.
    - The feature matrix `X` (11 features) is created, and polynomial features (`X_poly`, 77 features) are generated using `PolynomialFeatures(degree=2, include_bias=False)`.
    - Target variables (`accel`, `gear`, `steer`, `brake`) are extracted from the CSV.
2. **Model Training**:
    - Two `LinearRegression` instances are used:
        - `linreg_poly`: Trains `accel`, `steer`, and `brake` models on `X_poly`.
        - `linreg_raw`: Trains the `gear` model on `X`.
    - For each target:

        - `accel`, `steer`, `brake`: Fit `linreg_poly` with `X_poly` and the target (`y`).
        - `gear`: Fit `linreg_raw` with `X` and the target.
3. **Model Saving**:
    - Models are saved as `.sav` files in `../training/models/`:
        - `acc_controller.sav`, `steering_controller.sav`, `brake_controller.sav`, `gear_controller.sav`.
    - Each saved model is loaded and verified to confirm the expected number of input features.
    - Absolute paths are printed for debugging.
4. **Output**:
    - The script prints training progress, feature counts, and model expectations, ensuring transparency and error detection.

### Training Data

The dataset (`lancer oval2.csv`) contains driving data with columns:

- `track_1`, `track_17`, `track_9`, `trackPos`, `angle`, `gear`, `rpm`, `speedX`, `brake`, `accel`, `steer`.
- Derived features (`prev_gear`, `prev_brake`, `abs_diff_track_angle_pos`) are computed during preprocessing.

## Rationale for Training Approach

1. **Feature Selection**:
    - The 11 features were chosen to capture critical aspects of the car's state (track position, orientation, speed, and control history) relevant to driving decisions.
    - Derived features like `abs_diff_track_angle_pos` enhance the model's ability to detect misalignment between the car and the track.
2. **Polynomial Features for Most Models**:
    - `accel`, `steer`, and `brake` require complex decision boundaries due to the dynamic nature of track navigation (e.g., sharp turns require coordinated steering and braking).
    - Polynomial features enable the model to learn these interactions without requiring a more complex model architecture.
3. **Raw Features for Gear**:

    - Gear shifting is a simpler decision, primarily driven by RPM and speed, which can be modeled effectively with linear relationships.
    - Using raw features avoids unnecessary complexity.
4. **Separate Models**:
    - Training separate models for each control action allows specialization, as each action depends on different aspects of the input data.
    - This modular approach simplifies debugging and model tuning.
5. **Supervised Learning**:
    - The CSV provides labeled data (sensor inputs mapped to control outputs), making supervised learning with regression a natural choice.
    - Polynomial regression balances model complexity and performance, suitable for the real-time constraints of TORCS.

## Conclusion

The Autonomous Racing Car Project successfully implements an autonomous driving system for the TORCS simulator using polynomial regression models. The system processes real-time sensor data, predicts control actions, and navigates the track effectively. The training process leverages historical data to build robust models, with polynomial features capturing nonlinear relationships for acceleration, steering, and braking, and raw features sufficing for gear shifting. The modular design and clear separation of training and inference ensure reliability and maintainability. Future improvements could include experimenting with more complex models (e.g., neural networks) or incorporating reinforcement learning to adapt to different tracks dynamically.