Autonomous Vehicle - Final Paper

James Derrod
jderrod@oxy.edu
Occidental College

1 Problem Statement

This project addresses the challenge of developing and testing autonomous driving systems through simulation-based reinforcement learning. This autonomous driving problem requires an agent to control a car to navigate randomly generated racetracks while making real-time decisions about steering, acceleration, and braking, given input about their environment. The agent aims to develop a policy which can maximize the reward it earns within an episode. To address this challenge, a 2D simulated environment was used which creates a controlled testing ground where an agent can safely learn driving behaviors through trial and error, without the high costs and safety risks associated with real-world vehicle testing.

Autonomous driving technology has the potential to significantly improve transportation safety and accessibility. The World Health Organization reports that approximately 1.2 million people die each year from road traffic accidents[4], human error being a major contributing factor. By developing more sophisticated autonomous driving systems, we can work towards reducing these accidents. Furthermore, simulation-based training environments open up the possibility of democratizing autonomous vehicle research, allowing more researchers and institutions to contribute to the advancement of this technology.

Value proposition: using reinforcement learning for complex driving tasks

2 Technical Background

Reinforcement learning, vehicle physics simulation, and procedural track generation form the foundation of this autonomous driving project. Each component plays a vital role in creating an environment where an AI agent can learn to navigate racing tracks efficiently.

2.1 Reinforcement Learning and Markov Decision Process

Reinforcement learning operates on the principle of learning through interaction and feedback, mathematically formalized through a Markov Decision Process (MDP)[3]. This framework breaks down complex tasks like autonomous driving into components that a computer can process and learn from:

- Agent and Environment: The agent (controlling the car) interacts with the environment (the race track) by observing its state and taking actions.
- State Space (**S**): Represents what the agent can observe about its environment. In the driving scenario, this includes the car's speed, position on the track, distance from track boundaries, and information about upcoming turns.
- Action Space (**A**): Describes the possible actions an agent can take. The car has three continuous actions: steering (turning left or right), acceleration (speeding up), and braking (slowing down), each applicable with varying intensity.
- Transition Function (**P**): Defines how actions change the current state. In a physics-based environment, this represents how the car's position and speed change when the agent applies controls. This is handled by Unity's physics system.
- Reward Function (**R**): Provides feedback on the agent's performance. The agent receives positive rewards for staying on track and progressing or completing laps, while receiving penalties for actions like hitting track boundaries or driving inefficiently.

Unlike traditional rule-based programming, this framework enables the agent to discover optimal driving strategies through trial and error. The agent develops a policy, otherwise known as a strategy for choosing actions by maximizing its cumulative rewards over time. This approach allows the agent to learn complex behaviors without explicitly programming rules for every possible situation.

2.2 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is the reinforcement learning algorithm used in this project. PPO improves upon traditional policy gradient methods by implementing a clipped objective function that prevents large policy updates, which can destabilize training[2]. The algorithm works by:

- Collecting data from the current policy as the agent interacts with the environment
- Computing the ratio between new and old policies
- Clipping this ratio to limit the size of policy updates
- Optimizing a surrogate objective function that balances exploration and exploitation

This approach makes PPO particularly suitable for continuous control tasks like autonomous driving, as it provides stable learning while being computationally efficient. The algorithm's conservative update strategy helps prevent catastrophic performance drops during training, which is crucial when learning complex driving behaviors.

2.3 Vehicle Physics in 2D

This project implements a 2D vehicle physics model that captures key driving dynamics:

- Forward Force: Acceleration in the direction the car is facing.
- Turning Force: Rotation of the car based on steering input.
- Friction: Resistance to movement that affects the car's ability to turn (drift).
- Momentum: The car's tendency to keep moving in its current direction.

These physics elements provide realistic vehicle behavior while maintaining computational efficiency, creating a balance between accuracy and learning performance.

2.4 Procedural Track Generation

Procedural track generation is based on spline mathematics, a fundamental concept in computational geometry. Splines are piecewise polynomial functions that create smooth curves by interpolating between control points. The key mathematical components include:

- Bézier curves: Mathematical curves governed by control points that define their shape and curvature
- Interpolation: The process of creating continuous curves between discrete points while maintaining smoothness
- Tangent Vectors: Directional guides at control points that influence curve smoothness and shape

In the context of race track generation, these mathematical principles ensure:

- Continuous curvature between track segments
- Physical feasibility of the racing surface
- Controllable geometric properties (such as minimum turn radius)

Understanding these foundations is crucial because they determine the way in which the agent is trained, as well as the drivability of the generated tracks, and directly impact the learning environment's quality for autonomous agents.

3 Prior Work

Research in autonomous driving and reinforcement learning can be grouped into three main areas that inform this project: commercial autonomous driving systems, reinforcement learning in vehicle control, and simulation-based training approaches.

3.1 Commercial Autonomous Systems

Tesla's development of Autopilot and Full Self-Driving (FSD) systems demonstrates the practical challenges of autonomous driving. While Tesla primarily uses supervised learning and computer vision approaches rather than reinforcement learning, their work highlights the importance of extensive testing and iterative development in controlled environments before deployment. This aligns with the project's simulation-first approach to autonomous vehicle development.

3.2 Reinforcement Learning in Vehicle Control

Recent advances in reinforcement learning for vehicle control have established several key approaches:

- Deep Q-Networks (DQN) have shown promise in vehicle control tasks, particularly for discrete action spaces. Wang et al. demonstrated that DQN-based systems can effectively handle complex driving scenarios while maintaining computational efficiency [**10492530**]. This project builds upon this work by extending these principles to continuous action spaces. item Kiran et al. conducted a comprehensive survey of deep reinforcement learning in autonomous driving, establishing a taxonomy of driving tasks and their corresponding algorithmic approaches [**9351818**]. Their work informed the choice of PPO (Proximal Policy Optimization) algorithm and state space design.
- Wang et al.'s research on highway driving scenarios showed how reinforcement learning can handle high-speed decision-making tasks [**9190040**]. While this project focuses on racing scenarios rather than highway driving, we adopt similar principles for handling high-speed navigation and trajectory planning.

3.3 Simulation-Based Training

Simulation environments have become increasingly important in autonomous vehicle development, offering sev-

eral advantages over real-world training:

- Safety: Allows for exploration of dangerous scenarios without real-world risk
- Speed: Enables rapid iteration and testing of different approaches
- Cost: Eliminates the need for expensive physical prototypes during initial development

This project builds upon these previous works while focusing specifically on the challenge of autonomous racing through reinforcement learning in a customized Unity simulation environment. This approach allows us to combine the benefits of simulation-based training with the adaptability of reinforcement learning algorithms.

4 Methods

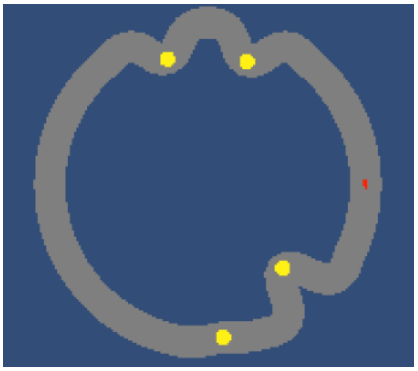
4.1 Track Generation System

The track generation system employs Unity's spline system for creating varied racing environments. The system generates tracks using three primary components: spline-based path generation, turn placement, and difficulty adjustment. The base track is constructed using a configurable number of control points (default=25) arranged in a circular pattern. Each control point influences the shape of the track through Bézier curve interpolation, ensuring smooth transitions between track segments. Track width is maintained as a constant parameter throughout the course, defining the drivable area's boundaries.

The turn system introduces complexity through three key parameters:

- Starting Turns: Initial number of turns (default=3)
- Maximum Turns: Upper limit for turn count (default=6)
- Turn Intensity: Controls turn sharpness (default=3.0)

Turns are placed along the track with enforced minimum spacing to prevent impossible sequences. Each turn modifies the base circular path by adjusting control points, creating natural deviations in the track's curvature. The difficulty progression system adapts track complexity based on agent performance. After each training episode, the system records whether the agent successfully completed the track. When the agent achieves a 75% completion rate over the last 100 episodes, the system increases track complexity by adding additional turns, up to the maximum limit. This adaptive approach ensures the agent faces increasingly challenging scenarios only after demonstrating competence at the current difficulty level.



4.2 Agent Design

The autonomous racing agent integrates sensor data, state observations, continuous actions, and a structured reward system to learn optimal racing behavior.

4.2.1 Sensor System

The agent perceives its environment through five raycasts positioned strategically around the vehicle: two side-facing rays for lateral distance detection, two 45-degree forward-angled rays for upcoming obstacle detection, and one forward-facing ray. Each raycast returns a normalized distance value between 0 and 1, representing the distance to track boundaries. This design provides the agent with essential proximity information while maintaining computational efficiency.

4.2.2 State Observation Space

The agent's observation space combines multiple data streams to provide comprehensive track awareness:

- Vehicle dynamics: Current velocity (x, y components), angular velocity, and time since last checkpoint
- Track progress: Distance to next checkpoint, current progress along spline (normalized position)
- Forward planning: Five look-ahead points on the track, providing advance information about upcoming track segments
- Turn information: Distance and direction to the next turn point, enabling anticipation of challenging track sections

4.2.3 Action Space

The agent operates with a continuous action space consisting of three parameters:

- Steering: Continuous value between -1 (full left) and 1 (full right)

- Acceleration: Continuous value between 0 and 1 (no acceleration -¿ full acceleration)
- Braking: Continuous value between 0 and 1 (no braking power -¿ full braking power)

These actions interact with Unity's physics system to produce realistic vehicle behavior, including momentum and drift effects.

4.2.4 Reward Structure

The reward system encourages both progression and skillful driving: Progressive Rewards:

- +1 for reaching each new spline point
- +3 for successfully completing a turn
- +10 for completing a full lap

Penalties:

- -5 for collision with track boundaries
- -8 for timeout (failing to progress)

This reward structure was designed to prioritize continuous forward progress while encouraging the completion of challenging track segments. The higher penalty for timeouts compared to collisions ensures the agent maintains forward momentum rather than becoming overly cautious.

4.3 Training Implementation

4.3.1 PPO Configuration

The agent was trained using Proximal Policy Optimization (PPO) with the following architecture and parameters: Network Configuration:

- Two hidden layers with 256 units each
- Normalized input and hidden states
- Simple visual encoding type

Key Hyperparameters:

- Batch size: 2048
- Buffer size: 20480
- Learning rate: 0.0003 (linear schedule)
- Gamma (discount factor): 0.99
- Lambda: 0.95
- Number of epochs: 4

Each training session ran for approximately 20-24 hours, accumulating 5-15 million steps per run.

4.3.2 Curriculum Learning

The training process implemented a performance-based curriculum:

- Evaluation Window: Performance measured over 100 episodes
- Success Threshold: 75% lap completion rate required for difficulty increase
- Progression: Incrementally increased track complexity through turn count and intensity

4.3.3 Evolution of Training Approach

The training process evolved through three major iterations: First Iteration:

- Basic reward structure focused on checkpoint completion
- Agent achieved sporadic success on simple tracks with three turns
- Identified need for more structured rewards

Second Iteration:

- Added lap completion rewards
- Achieved significantly higher mean rewards
- Agent demonstrated ability to complete multiple consecutive laps

Third Iteration:

- Introduced turn-specific rewards and observations
- Implemented adaptive difficulty progression
- Encountered challenges with increased complexity, suggesting potential oversaturation of the observation space or need for hyperparameter adjustment

Each iteration provided insights that informed subsequent training approaches, though the increased complexity in the final iteration revealed potential limitations in the current architecture that warrant further investigation.

5 Evaluation Metrics

The mean reward per episode served as the primary quantitative measure, reflecting the agent's overall effectiveness in completing track objectives. During successful training runs, mean rewards reached into the hundreds, indicating multiple checkpoint completions and successful lap navigation.

Episode length, measured in steps, provided insight into the agent's efficiency and consistency. Longer episodes typically indicated successful navigation, while shorter episodes often resulted from early collisions or timeout failures. Successful agents maintained episode lengths sufficient to complete multiple laps, approximately 500-1000 steps per lap depending on track complexity.

The lap completion rate proved particularly valuable for assessing agent reliability. This metric tracked the percentage of episodes where the agent successfully completed a

full circuit of the track. In the most successful iteration (second training run), the agent achieved consistent lap completions, maintaining the required 75% success rate over 100-episode windows.

Turn completion success rate, introduced in the third iteration, specifically measured the agent's ability to navigate challenging track segments. Each successful turn completion earned a +3 reward, allowing us to track the agent's proficiency at handling these critical track features.

5.0.1 Training Progress Metrics

We monitored training progress through TensorBoard, tracking reward trends and learning stability. Key indicators included:

- Running average of episode rewards
- Frequency of successful lap completions
- Distribution of episode lengths
- Rate of premature episode terminations

These metrics revealed clear patterns in the agent's learning progression, including both steady improvements and plateau periods where performance stabilized or occasionally degraded.

5.1 Comparative Analysis

5.1.1 Across Training Iterations

The agent's capabilities evolved significantly across three training iterations:

- First Iteration:
  - Achieved basic track navigation with sporadic success on tracks with three turns
  - Mean rewards remained relatively low, indicating inconsistent performance
  - Limited ability to handle track variations or complete full laps consistently
- Second Iteration:
  - Introduced lap completion rewards (+10), leading to dramatically improved performance
  - Mean rewards reached hundreds, indicating multiple lap completions
  - Demonstrated consistent ability to navigate tracks with 3-4 turns
  - Showed marked improvement in maintaining continuous forward progress
- Third Iteration:
  - Added turn awareness and adaptive difficulty
  - Initially showed promise with turn-specific learning
  - Performance plateaued when facing increased complexity
  - Suggested possible limitations in the current architecture when handling multiple objectives

5.1.2 Performance at Different Difficulty Levels

Track difficulty variations revealed clear performance patterns:

- Turn Count Impact: Performance decreased notably as turns increased from 3 to 6
- Turn Intensity: Higher turn intensity (¿3.0) significantly reduced completion rates
- Track Width: Wider tracks generally resulted in higher completion rates

5.2 Failure Analysis

5.2.1 Common Failure Modes

Analysis of unsuccessful episodes revealed several recurring patterns: Timeout Scenarios:

- Most common in complex track sections with multiple consecutive turns
- Often occurred when the agent became "stuck" trying to navigate tight turns
- Indicated potential issues with the agent's long-term planning capabilities

Collision Patterns:

- Frequently occurred at turn entry points
- More common with higher turn intensity settings
- Often resulted from excessive speed entering turns

Training Plateaus:

- Occurred consistently when complexity increased too rapidly
- Suggested limitations in the agent's ability to handle multiple objectives simultaneously
- Indicated potential issues with the reward structure balancing

5.2.2 Performance Limitations

Analysis revealed several key limitations: Complexity Threshold:

- Agent performance degraded significantly beyond 4 turns per track
- Struggled to maintain consistent performance when multiple learning objectives were introduced
- Showed difficulty generalizing to more complex track layouts

Behavioral Inconsistencies:

- Occasional erratic behavior even after successful training
- Some inconsistent performance across similar track configurations.

Training Stability:

- Third iteration showed signs of catastrophic forgetting.
- Increased complexity led to unstable learning patterns.

These limitations point to several areas for future improvement, particularly in the agent's ability to handle increased track complexity while maintaining stable performance.

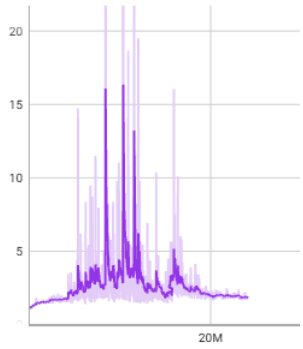
6 Results and Discussion

Evaluation of autonomous racing agent across multiple training iterations revealed both significant achievements and notable limitations in reinforcement learning for autonomous racing. The findings span three key areas:

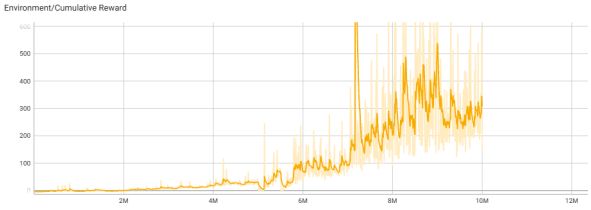
6.1 Training Performance Analysis

6.1.1 Learning Progression

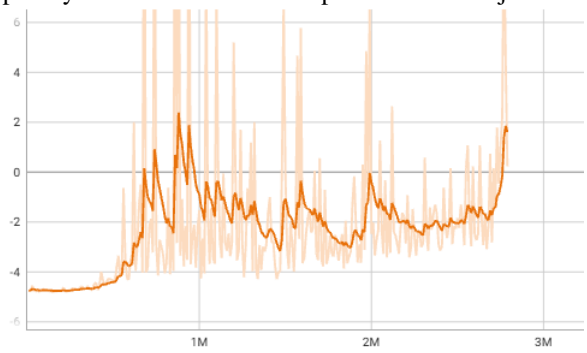
- Reward accumulation showed distinct patterns across iterations, with initial slow progress followed by periods of rapid improvement
- First iteration achieved basic navigation but with inconsistent rewards, rarely exceeding single-lap completion



- Second iteration demonstrated significant improvement, with mean rewards reaching hundreds, indicating multiple lap completions]



- Third iteration initially showed promise but struggled with the increased complexity of turn-specific objectives



6.1.2 Behavioral Analysis

- Agent developed efficient navigation strategies for straightaways and basic turns
- Learning emerged in stages: first maintaining track position, then optimizing speed, and finally attempting turn optimization
- Vehicle control became more refined over time, with smoother acceleration and braking patterns
- Turn handling showed adaptation to different angles and intensities, though performance decreased with turn complexity

6.2 Performance Benchmarks

6.2.1 Track Navigation

- Second iteration achieved the best results, demonstrating consistent multi-lap completions

6.3 System Limitations and Challenges

6.3.1 Technical Constraints

- Training sessions were quite long.
- Five-raycast sensor system showed limitations in complex scenarios
- Training required 5-15 million steps per iteration to achieve stable performance

6.3.2 Learning Challenges

- Performance degraded significantly with tracks containing more than 4 turns
- Agent struggled to balance multiple objectives when turn-specific rewards were introduced
- Difficulty generalizing learned behaviors to substantially different track layouts

- Evidence of catastrophic forgetting during third iteration when complexity increased

6.4 Future Improvements

6.4.1 Architectural Enhancements

- Implement additional sensor rays for better environmental awareness
- Explore deeper neural network architectures to handle increased complexity
- Investigate alternative PPO configurations to improve training stability, or alternative learning algorithms.
- Consider implementing a memory system to better handle long-term dependencies

6.4.2 Training Refinements

- Develop more granular curriculum learning stages
- Implement dynamic reward scaling based on task difficulty
- Explore transfer learning to preserve successful behaviors while learning new skills
- Consider implementing a hybrid reward system that better balances multiple objectives

These results demonstrate that while reinforcement learning can successfully teach an agent basic autonomous racing skills, significant challenges exist in scaling to more complex scenarios. The success of the second iteration suggests that carefully balanced reward structures and appropriate complexity progression are crucial for effective learning.

7 Ethical Considerations

The development of autonomous racing systems raises key ethical considerations that extend beyond simulation to real-world applications. This project addresses these concerns in two main areas:

- **Safety in Development**: Simulation-based training provides a risk-free environment for testing autonomous driving behaviors that would be dangerous to explore with physical vehicles. This approach allows for extensive testing of edge cases and aggressive maneuvers without endangering people or property.
- **System Transparency**: Reinforcement learning agents can develop effective but difficult-to-interpret decision-making processes. By documenting the agent's success and failure modes[**AVPowerInequity**], we contribute to understanding reinforcement learning limitations in autonomous systems and highlight areas where improved transparency is needed.

Future work should focus on incorporating explicit safety constraints and developing more interpretable decision-making processes[1] to ensure that advances in autonomous vehicles maintain safety as a priority.

8 Replication Instructions

To replicate this project, go to https://github.com/jderrod/JamesDerrodCompsProjectCode and clone the repository. Open up in unity, and go to the "project" scene.

To run, ensure the car agent script attatched to the car is in default mode, not heuristic or inference. Insert the trained model by dropping in the .onnx file from the results folder "1" or "5". Choose the .onnx file with the highest number and run the program. To train headlessly, running within the directory:

Start the virtual environment with: "MLvenv", then, "mlagents-learn Assets/training$_c onfig.yaml - -env =$ "$Builds/CompsProject2.exe$" $- -run - id =$ $new_r un_i d - -no - graphics$"

9 Code Architecture Overview

This autonomous racing system consists of two primary components: the TrackGenerator and CarAgent classes, which handle environment generation and agent behavior respectively.

9.1 TrackGenerator Class

The track generation system manages procedural track creation and difficulty progression:

- **Track Creation Pipeline**:
  - GenerateRandomSpline(): Creates base track shape
  - GenerateTrackMesh(): Converts spline to drivable surface
  - PlaceTurnMarkers(): Handles turn point identification

  Copy
- **Difficulty Management**:
  - RecordLapCompletion(): Tracks agent performance
  - GenerateRandomTurnIndices(): Controls turn placement
  - Dynamic difficulty adjustment based on completion rates

9.2 CarAgent Class

The agent implementation extends Unity's ML-Agents framework:

- **Core Components**:
  - CollectObservations(): Gathers environment state
  - OnActionReceived(): Processes agent decisions
  - GetTrackEdgeSensorReadings(): Manages sensor system

  Copy
- **Physics Integration**:
  - ApplyCarControls(): Handles vehicle physics
  - IsWithinTrackBounds(): Manages boundary detection
  - CheckProgressTimer(): Tracks agent progress

Extension points for future development include:

- Additional sensor types in GetTrackEdgeSensorReadings()
- New reward mechanisms in CheckProgressTimer()
- Enhanced track generation features in GenerateRandomSpline()
- Extended difficulty progression logic in RecordLapCompletion()

References

[1] David Danks, Alex John London. *Algorithm Bias in Autonomous Systems*. 2017. URL: https://www.cmu.edu/dietrich/philosophy/docs/london/IJCAI17-AlgorithmicBias-Distrib.pdf.

[2] Schulman, John et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[3] Wikipedia. *Markov Decision Process*. 2024. URL: https://en.wikipedia.org/wiki/Markov_decision_process.

[4] World Health Organization. *Road traffic injuries*. Accessed: 2024-03-15. 2023. URL: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries.