Dynamic Speed Optimization for Autonomous Navigation using Reinforcement Learning

Elton Roque Lemos
50510481
The State University of New York, at Buffalo
Buffalo, New York
eltonroq@buffalo.edu

Yamini Ramesh
50540009
The State University of New York, at Buffalo
Buffalo, New York
yaminira@buffalo.edu

***Abstract*—This research introduces a dynamic speed optimization system for autonomous navigation, leveraging reinforcement learning within a deep learning framework. Through a series of RL-based cases, we systematically investigate modifications aimed at enhancing speed optimization in robotic pathing algorithms. Our findings showcase notable enhancements in safety, lap times, and adaptability to diverse environmental conditions. By synergizing the capabilities of RL and DL, our approach provides a holistic solution to the challenges encountered in high speed navigation, thus advancing the field of autonomous vehicle technology.**

***Index Terms*—autonomous vehicles, safety, lap time optimization**

I. Introduction

Autonomous vehicles represent a significant technological advancement with the potential to revolutionize transportation systems worldwide. However, ensuring their safe and efficient operation poses formidable challenges, particularly in high speed navigation scenarios. The future of autonomous vehicles is an ambitious era of safe and comfortable transportation [1]. Traditional pathing algorithms such as A*, RRT, and gap follow, while effective in many respects, encounter difficulties when vehicles operate at high speeds [2]. These challenges include computational inefficiencies, resulting in delayed path computation and potential safety hazards.

The limitations of traditional pathing algorithms become especially pronounced when vehicles navigate dynamic and complex environments, such as race tracks, where quick decision-making is paramount. Without an effective speed optimization mechanism, autonomous vehicles may experience sub-optimal lap times, increased fuel consumption, and heightened safety risks [3].

To address these challenges, we propose a novel approach that integrates reinforcement learning (RL) within a deep learning (DL) framework to dynamically optimize vehicle speed during navigation. By incorporating RL-based speed optimization into robotic pathing algorithms, our research aims to enhance the safety, efficiency, and adaptability of autonomous vehicles in high-speed scenarios.

In this paper, we present the results of a comprehensive study comprising nine iterative cases, each focusing on refining and improving the RL-based speed optimization system. Through empirical experimentation and analysis, we demonstrate the effectiveness of our approach in overcoming the challenges posed by high-speed navigation, thereby advancing the state-of-the-art in autonomous vehicle technology.

A. Literature Review

The pursuit of optimizing speed for autonomous vehicles through reinforcement learning (RL) has garnered significant attention in recent years, driven by the imperative to enhance both safety and efficiency in high-speed navigation scenarios. Autonomous vehicles have witnessed remarkable advancements since the inception of radio-controlled vehicles in the 1920s. Subsequent decades saw the evolution of autonomous electric cars with embedded circuits in roads. By the 1960s, vision-guided autonomous vehicles emerged as a major milestone, laying the groundwork for modern-day navigation technologies.

Researchers Siegwart and Nourbakhsh (2004) highlighted the fundamental principles of autonomous robotics, underscoring the importance of sensor data processing and decision-making algorithms [4]. Building upon this foundation, Schwarting et al. (2018) explored planning and decision making strategies specifically tailored for autonomous vehicles, emphasizing the integration of control and robotics methodologies.

The intersection of RL and deep learning (DL) has emerged as a promising paradigm for speed optimization in autonomous vehicles. Leveraging RL algorithms, researchers have developed adaptive speed control mechanisms capable of dynamically adjusting vehicle velocity based on real-time environmental cues [5]. By optimizing speed profiles using RL, autonomous vehicles can navigate complex environments more effectively, mitigating safety risks while optimizing travel times.

However, challenges persist in developing RL-based speed optimization systems that seamlessly integrate with existing pathing algorithms. Computational complexity, training time, and model robustness are among the key considerations in designing effective RL frameworks for speed optimization in autonomous vehicles [6].

As the automotive industry marches towards fully autonomous vehicles, the integration of RL-based speed opti-

mization represents a critical step towards achieving safer, more efficient transportation systems. By harnessing the power of RL and DL, researchers aim to propel autonomous vehicle technology into an era of unprecedented safety and performance.

B. Objectives and Problem Statement

The primary objective of this research is to develop a robust speed optimization system for autonomous vehicles using reinforcement learning (RL) in conjunction with deep learning (DL) techniques. Specifically, the objectives include:

1. Designing an RL-based framework capable of dynamically adjusting vehicle speed based on real-time environmental cues.

2. Integrating the RL framework with existing pathing algorithms to optimize speed profiles while ensuring safe navigation.

3. Evaluating the performance of the proposed speed optimization system in terms of safety, efficiency, and adaptability to diverse driving scenarios.

4. Investigating the computational efficiency and scalability of the RL-based speed optimization system for real-world deployment.

Autonomous vehicles face significant challenges in optimizing speed while navigating complex environments. Traditional pathing algorithms such as A*, RRT, and gap follow are effective but often fail to compute safe paths at high speeds. Conversely, running vehicles at slower speeds compromises efficiency and increases travel times. To address these challenges, there is a need for a speed optimization system that can dynamically adjust vehicle velocity in real-time, ensuring both safety and efficiency. This research aims to develop such a system by leveraging RL techniques to learn optimal speed profiles based on environmental inputs. By integrating RL with existing pathing algorithms, the system seeks to strike a balance between safety and speed, enabling autonomous vehicles to navigate diverse environments with agility and precision.

II. Autonomous Vehicle Simulation

We use an autonomous vehicle simulator that was built by our fellow classmates of Robotics department called the "Drones Lab F1-10th Simulator", which works in real-time considering the environmental effects. The audubon-unity package is a simulator built using Unity, specifically designed for testing and development in the field of autonomous vehicle research. This simulator provides a realistic environment with various race tracks, a controllable vehicle, and simulated sensor data, including lidar scans and odometry information. The simulator incorporates real-time environmental effects such as gravity and friction to accurately mimic driving conditions. To train an autonomous vehicle system, you need to produce an environment that is as closely related to what the real car would see on the road. The key parts of our simulation environment are:

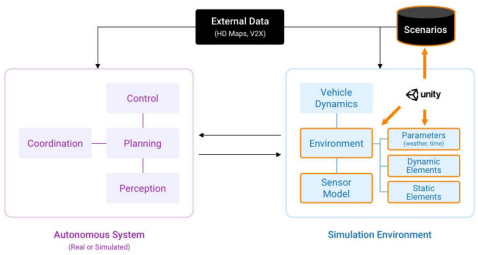

Fig. 1. Simulation Architecture of Unity for Evaluating Autonomous Systems in a Virtual Environment

Vehicle dynamics: how the car behaves physically, such as friction with the asphalt.

Environment: This part comprises two sub-categories:

Static elements, such as the obstacle blocks. Dynamic elements, such as dynamically appearing obstacle blocks, that provide variations within our scenario and allow us to create scenarios that can be used to validate or collect data for our vehicle.

The conjunction of these varied environment factors is what allows us to produce edge cases that are rare in reality.

Sensor model: The simulation scenarios need to be taken in by the autonomous system via a sensor model, such as a LiDAR sensor, camera, or radar. It has to be physically accurate to the point that the algorithm relying on this information can behave in the synthetic environment as well as it will behave in reality.

A. Functionality

The audubon-unity simulator allows users to simulate autonomous vehicle scenarios by controlling a virtual car and receiving sensor data feedback. By coding in ROS (Robot Operating System), users can interact with the simulated environment, control the vehicle's speed and trajectory, and implement various algorithms for navigation and obstacle avoidance.

Simulator Environment: The audubon-unity simulator is built using Unity, providing a visually realistic environment with detailed race tracks, dynamic lighting, and terrain features.

Vehicle Control: Users can control a virtual car within the simulator, adjusting its speed, direction, and other parameters using ROS commands.

Sensor Simulation: The simulator generates simulated sensor data, including lidar scans and odometry information, allowing users to develop and test perception algorithms for autonomous navigation.

ROS Integration: The simulator is integrated with ROS, enabling users to interact with the simulated environment using ROS topics, services, and actions.

Customization: Users can customize the simulator environment and vehicle characteristics, allowing for experimentation with different scenarios and vehicle configurations.

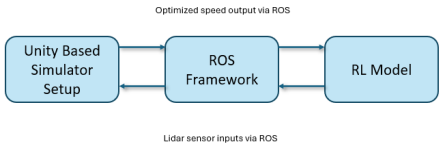

Fig. 2. Data Flow Diagram of Lidar Sensor Input and Speed Optimization Output in the Unity-ROS-RL Model Framework

B. Technology Stack

Unity: Used for building the simulator environment and providing realistic graphics and physics simulations.

ROS: Provides the framework for controlling the simulated vehicle, processing sensor data, and implementing algorithms.

CMake: Manages the build process and dependencies for the audubon-unity package.

III. Implementation

A. Project Architecture

Our project architecture encompasses a comprehensive setup designed to facilitate the training and optimization of autonomous vehicle speed using reinforcement learning (RL) techniques. Below is a detailed overview of the various components and their interactions within the architecture:

1. Simulator Setup: We leveraged an advanced Unity-based simulator integrated with Robot Operating System (ROS) functionality. This simulator provided a realistic environment for the autonomous vehicle to operate in, complete with dynamic physics, environmental effects (such as gravity and friction), and sensor data simulation. The simulator included multiple race tracks, each presenting unique challenges and characteristics, allowing for diverse training scenarios. This block represents the Unity based simulator with ROS integration. It provides the virtual environment for the autonomous vehicle to operate in.

2. Training Process: RL training was conducted within the simulator environment, enabling the autonomous vehicle to learn and adapt its speed control policies in real-time. During training runs, the vehicle continuously interacted with the simulator, receiving observations from the environment (such as lidar scan data) and taking actions based on its learned policy. The RL algorithm iteratively adjusted its policy based on the received rewards, optimizing for both lap time performance and safety. This block represents the core RL training loop happening inside the simulator. The vehicle interacts with the environment, receives observations (lidar data), and takes actions (speed control) based on its learned policy.

3. Neural Network Design: A key component of our architecture was the design of the neural network (NN) responsible for processing observations and determining optimal speed control actions [7]. The NN consisted of fully connected layers with varying architectures, tailored to handle the complexity of lidar scan data and previous action inputs. We experimented with different layer sizes and depths to optimize learning performance and decision making accuracy. This block represents the neural network that receives pre-processed observations (lidar data and previous action) and outputs the optimal speed control action.

4. Observation Processing: Lidar scan data, comprising 1080 points representing the vehicle's surroundings, served as the primary input to the NN. Additionally, the previous action taken by the vehicle was included as part of the observation, providing context for the current decision-making process. Observations were pre-processed and fed into the NN for feature extraction and action selection. This block represents the pre-processing steps applied to the raw lidar scan data before feeding it into the NN.

5. Action Selection: To facilitate smooth and precise speed control, we discretized the available actions into distinct speed levels. The action space represented various speed settings, allowing the vehicle to accelerate, decelerate, or maintain cruise speed within predefined bounds. This discretization simplified the decision-making process for the NN, enabling it to learn effective speed control policies. This block signifies the NN's decision-making process, where it selects the optimal speed control action based on the processed observations and learned policy.

6. Reward System: A carefully balanced reward structure incentivized safe and efficient driving behavior while penalizing violations and unsafe actions. Rewards were assigned based on various factors, including lap time performance, completion status, and collision avoidance [8]. By adjusting reward values, we aimed to strike a balance between optimizing lap times and ensuring safe driving practices. This block represents the system that assigns rewards or penalties based on the vehicle's performance (lap time, completion, safety) during training runs.

7. Performance Evaluation: Throughout the training process, we continuously evaluated the performance of the autonomous vehicle to assess the effectiveness of the speed control policies learned by the neural network. Performance metrics included lap times, completion rates, collision frequency, and overall driving efficiency. We conducted extensive testing on diverse race tracks within the simulator to ensure robustness and generalization of the learned policies across different environments. Performance evaluation results were analyzed iteratively, guiding further adjustments and refinements to the training process and neural network architecture. By rigorously evaluating performance metrics, we aimed to achieve a balance between competitive lap times and safe driving behavior, ultimately optimizing the autonomous vehicle's speed control capabilities. This block represents the overall evaluation of the trained model's performance, considering factors like lap times and safety metrics.

The flow of information between the blocks are as follows: Pre-processed observations and previous actions are fed into the NN for action selection. The selected action is then taken in the simulator, and the resulting observations and rewards are

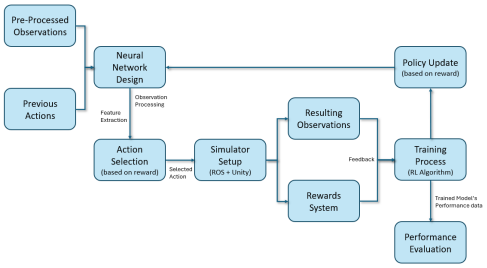

Fig. 3. Comprehensive Block Diagram of the Autonomous Vehicle Speed Optimization Project Architecture

fed back into the training process to update the NN's policy. The performance evaluation happens outside the main training loop but utilizes the trained model's performance data.

Our project architecture provided a robust framework for training and optimizing autonomous vehicle speed control using RL techniques. By integrating advanced simulation capabilities, neural network processing, and reward-driven learning, we developed a sophisticated system capable of achieving competitive lap times while maintaining safety standards on challenging race tracks. This architecture lays the foundation for further advancements in autonomous vehicle control and optimization, driving innovation in the field of autonomous transportation.

B. DQN Modifications for Real-Time Autonomous Vehicle Speed Optimization

Our implementation of Deep Q-Network (DQN) was adapted to better accommodate the demands of real-time autonomous vehicle speed optimization within a simulated environment. These modifications addressed the need for continuous operation without interruption, balanced reward mechanisms, and effective staged training.[9]

1. Continuous Operation:

Standard DQN vs. Modified DQN: In traditional DQN implementations, the model typically stops after each test episode to receive rewards and update its policy. This episodic approach, while effective in many scenarios, was unsuitable for our real-time application where the vehicle needed to continuously operate without interruption.

End-of-Episode Training: To maintain continuous operation, we modified the DQN to defer training until the end of an entire lap. This way, the vehicle continuously receives real-time instructions throughout the lap, avoiding any interruptions that could lead to crashes due to delayed decision-making. Training and policy updates occur only after completing the lap, ensuring the vehicle's performance remains smooth and uninterrupted during the run.[10]

2. Reward Structuring and Curriculum Training:

Reward Based on Speed: Initially, our reward system incentivized the car's speed, providing higher rewards for higher speeds. However, this led to the vehicle prioritizing speed over safety, often accelerating when it should decelerate, particularly at turns or in collision-prone situations.

Overriding Negative Rewards: The speed-based reward system overwhelmed the negative rewards for crashes, resulting in reckless driving behavior. To counter this, we introduced Curriculum Training.

Curriculum Training:

Phase 1 - Racing Focus: The first phase of training focused on teaching the car to race by maximizing speed. During this phase, the model learned to accelerate and maintain high speeds on straight paths and gentle curves.

Phase 2 - Safety Focus: Once the model demonstrated proficiency in racing, we transitioned to a safety-focused phase. This phase emphasized cautious navigation of turns, appropriate deceleration, and crash avoidance, balancing the initial speed-focused training with safe driving practices.

Early Stopping: We employed early stopping to select a model that effectively balanced speed and safety. This technique allowed us to halt training at an optimal point, ensuring the model retained its learned safety behaviors while still achieving competitive lap times. Early stopping helped prevent over-fitting and ensured robust performance across various driving scenarios.

3. Independence from Steering Control:

Steering Control via PID Controller: Our neural network does not consider steering as a factor in its decision-making process for speed optimization. The steering is managed separately by a Proportional-Integral-Derivative (PID) controller, ensuring that the neural network's focus remains solely on speed control.

Unbiased Speed Optimization: By not integrating steering control into the neural network, we ensured that the speed optimization decisions are made independently of the steering angle. This separation allows the neural network to concentrate on optimizing speed based on lidar sensor data and other relevant inputs, without being biased by the steering adjustments.

IV. Evaluation

A. Evaluations Cases

The evaluation of our research involved assessing the performance and effectiveness of different modifications and enhancements applied to the autonomous vehicle speed control system. We conducted a series of experiments and analyses to measure the impact of each modification on key performance metrics and overall system behavior. Below, we outline the evaluation process and present the findings from each case study:

Case 1: Baseline Model

- Objective: Evaluate the performance of the baseline model with simple actions and reward structure.
- Metrics: Lap completion time, crash frequency, and overall lap completion rate.
- Rewards: 0 for every step. -100 for not completing. -50 for crash. 600 – time taken for completion of a lap.
- Terminated: When 2 laps are completed.
- Truncated: When the car crashes.
- NN Architecture:
  self.fc1 = nn.Linear(1081, 512)

self.fc2 = nn.Linear(512, 128)
self.fc3 = nn.Linear(128, 64)
self.fc4 = nn.Linear(64, 3)

- Findings: Observing from the Lidar scan (1080) and previous action (1) the baseline model struggled to learn effective braking behaviors before turns, resulting in sub-optimal lap times and frequent crashes.

Case 2: Harsher Penalties

- Objective: Assess the impact of harsher penalties on the learning process.
- Metrics: Lap completion time, crash frequency, and overall lap completion rate.
- Rewards: -1 for every step. -1000 for not completing. -500 for crash. 600 – time taken for completion of a lap.
- Terminated: When 2 laps are completed.
- Truncated: When the car crashes.
- NN Architecture:
  self.fc1 = nn.Linear(1080, 512)
  self.fc2 = nn.Linear(256, 128)
  self.fc3 = nn.Linear(128, 64)
  self.fc4 = nn.Linear(64, 3)
- Observing the Lidar scan (1080), while the model learned to brake more frequently, the effectiveness of braking was limited by inefficient braking logic. Lap times improved slightly, but crashes remained a significant issue.

Case 3: New Actions

- Objective: Introduce a new action space to enable more nuanced speed control.
- Metrics: Lap completion time, crash frequency, and overall lap completion rate.
- Actions: Discretized versions of actual output speed (35 actions).
- Rewards: -1 for every step. -1000 for not completing. -500 for crash. 600 – time taken for completion of a lap.
- Terminated: When 2 laps are completed.
- Truncated: When the car crashes.
- NN Architecture:
  self.fc1 = nn.Linear(1080, 512)
  self.fc2 = nn.Linear(256, 128)
  self.fc3 = nn.Linear(128, 64)
  self.fc4 = nn.Linear(64, 3)
- Findings: The introduction of discretized speed actions allowed for smoother speed adjustments, reducing the frequency of random stops. However, lap times did not improve significantly, indicating the need for further refinement.

Case 4: Scaled Down Rewards

- Objective: Adjust reward structure to encourage safer driving behaviors.
- Metrics: Lap completion time, crash frequency, and overall lap completion rate.
- Actions: Discretized versions of actual output speed (35 actions).
- Rewards: -1 for every step. -50 for not completing. -10 for crash. -60 for 3rd crash. 300 – 4*(time taken for completion of a lap).
- Terminated: When 2 laps are completed.
- Truncated: When the car crashes 3 times.
- NN Architecture:
  self.fc1 = nn.Linear(1080, 512)
  self.fc2 = nn.Linear(256, 128)
  self.fc3 = nn.Linear(128, 64)
  self.fc4 = nn.Linear(64, 3)
- Findings: While crashes were less penalized, the model struggled to differentiate between different turns, leading to inconsistent performance. Lap times showed minimal improvement compared to the baseline.

Case 5: New Neural Network

- Upgrade neural network architecture to improve learning capability.
- Metrics: Lap completion time, crash frequency, and overall lap completion rate.
- Actions: Discretized versions of actual output speed (35 actions).
- Rewards: -1 for every step. -50 for not completing. -10 for crash. -60 for 3rd crash. 300 – 4*(time taken for completion of a lap). 0.5* speed at every time step at episodes ¡1500. 0 at every time step for episodes 1500-2000(end of training).
- Terminated: When 2 laps are completed.
- Truncated: When the car crashes 3 times.
- NN Architecture:
  self.fc1 = nn.Linear(1080, 512)
  self.fc2 = nn.Linear(512, 512)
  self.fc3 = nn.Linear(512, 128)
  self.fc4 = nn.Linear(128, 35)
- Findings: The new neural network architecture enabled the model to achieve competitive lap times initially. However, performance degradation mid-training suggested potential over-fitting or instability issues.

Case 6: Reduced Learning Rate

- Objective: Investigate the impact of reducing the learning rate on model convergence and stability.
- Metrics: Lap completion time, crash frequency, and overall lap completion rate.
- Actions: Discretized versions of actual output speed (35 actions).
- Rewards: -1 for every step. -50 for not completing. -10 for crash. -60 for 3rd crash. 300 – 4*(time taken for completion of a lap). 0.5* speed at every time step at episodes ¡1500. 0 at every time step for episodes 1500-2000(end of training).
- Terminated: When 2 laps are completed.
- Truncated: When the car crashes 3 times.
- NN Architecture:
  self.fc1 = nn.Linear(1080, 512)
  self.fc2 = nn.Linear(512, 512)

self.fc3 = nn.Linear(512, 128)
self.fc4 = nn.Linear(128, 70)

- Findings: Lowering the learning rate improved model stability but did not address the core issue of the car staying stationary or flipping in extreme cases. Lap times remained inconsistent.

Case 7: Fine-Tuning with PID Control

- Objective: Implement PID control for smoother acceleration and deceleration.
- Metrics: Lap completion time, crash frequency, and overall lap completion rate.
- Actions: Discretized versions of actual output speed (35 actions).
- Rewards: (Current speed) for every step. -50 for not completing. -10 for crash. -60 for 3rd crash. 300 – 4*(time taken for completion of a lap). 0.5* speed at every time step at episodes ¡1500. 0 at every time step for episodes 1500-2000(end of training).
- Terminated: When 2 laps are completed.
- Truncated: When the car crashes 3 times.
- NN Architecture:
  self.fc1 = nn.Linear(1080, 512)
  self.fc2 = nn.Linear(512, 512)
  self.fc3 = nn.Linear(512, 128)
  self.fc4 = nn.Linear(128, 70)
- Findings: While PID control improved speed management, network glitches and simulator issues persisted, leading to erratic behavior. Lap times showed marginal improvement.

Case 8: Moved Training

- Objective: Adjust training process timing to address learning deficiencies.
- Metrics: Lap completion time, crash frequency, and overall lap completion rate.
- Actions: Discretized versions of actual output speed (35 actions).
- Rewards: (Current speed) for every step. -50 for not completing. -5speed for crash. -10speed for 3rd crash. 300 – 4*(time taken for completion of a lap). 0.5* speed at every time step at episodes ¡1500. 0 at every time step for episodes 1500-2000(end of training).
- Terminated: When 2 laps are completed.
- Truncated: When the car crashes 3 times.
- NN Architecture:
  self.fc1 = nn.Linear(1080, 512)
  self.fc2 = nn.Linear(512, 512)
  self.fc3 = nn.Linear(512, 128)
  self.fc4 = nn.Linear(128, 70)
- Findings: Despite training just before car reset, the model continued to struggle with turn anticipation and braking. Lap times did not show significant improvement.

Case 9: Improved NN and Rewards

- Objective: Enhance neural network architecture and reward structure for better convergence.
- Metrics: Lap completion time, crash frequency, and overall lap completion rate.
- Actions: Discretized versions of actual output speed (35 actions).
- Rewards: (Current speed) for every step. -500 for not completing. -50speed for crash. -100speed for 3rd crash. 240 – 4*(time taken for completion of a lap). 0.5* speed at every time step at episodes ¡1500. 0 at every time step for episodes 1500-2000(end of training).
- Terminated: When 2 laps are completed.
- Truncated: When the car crashes 3 times.
- NN Architecture:
  self.fc1 = nn.Linear(1080, 2048)
  self.fc2 = nn.Linear(2048, 512)
  self.fc3 = nn.Linear(512, 256)
  self.fc4 = nn.Linear(256, 90)
- Findings: The larger neural network architecture showed promise in handling more complex scenarios, but convergence remained challenging. Early stop implementation and additional observations may be necessary for improved performance.

Case 10: Implement CNN

- Objective: Utilize Convolution Neural Network (CNN) to improve learning and performance of speed optimization
- Metrics: Lap completion time, crash frequency, and overall lap completion rate.
- Actions: Discretized versions of actual output speed (35 actions). Car speed = action * 0.5.
- Rewards: (Current speed) for every step. -500 for not completing. -50 * speed for a crash. -100 * speed for the 3rd crash. 240 – 4 * (time taken for completion of a lap). 0.5 * speed at every time step.
- Terminated: When 2 laps are completed.
- Truncated: When the car crashes 3 times.
- NN Architecture:
  elf.cn = nn.Conv1d(4, 8, 8, 8)
  self.fc1 = nn.Linear(1080, 256)
  self.fc2 = nn.Linear(256, 128)
  self.fc4 = nn.Linear(128, 70)
- Notes about Implementation: Learning was done during the run itself.
  The car had 3 lifelines and used a PID-controlled acceleration function.
  Training happened just before a car reset was required. Observations were taken over 4 frames, stacked, and input into the neural network.
  A 1D convolution and pooling were applied to this input, followed by a reduced complexity dense neural network.
- Findings: The reward of 0.5 * speed at every time step caused the model to prioritize speed over safety, leading to frequent crashes.
  Removing the reward led to the model failing to reach the goal in time.
  To balance speed and safety, the model was first trained to prioritize speed. Once convergence was achieved, the

speed reward was removed, and training continued to emphasize safety.
Early stopping was used to prevent the model from over-learning safety and becoming too slow.

Case 11: Shape Rewards

- Objective: Shape rewards to balance speed and safety during training.
- Metrics: Lap completion time, crash frequency, and overall lap completion rate.
- Actions: Discretized versions of actual output speed (35 actions). Car speed = action * 0.5.
- Rewards: (Current speed) for every step -500 for not completing. -50 * speed for a crash. -100 * speed for the 3rd crash. 240 – 4 * (time taken for completion of a lap). 0.5 * speed at every time step for episodes ¡ 1500. 0 at every time step for episodes 1500-2000 (end of training)
- Terminated: When 2 laps are completed.
- Truncated: When the car crashes 3 times.
- NN Architecture:
  self.cn = nn.Conv1d(4, 8, 8, 8)
  self.fc1 = nn.Linear(1080, 256)
  self.fc2 = nn.Linear(256, 128)
  self.fc4 = nn.Linear(128, 70)
- Notes about Implementation: Learning was done during the run itself.
  The car had 3 lifelines and used a PID-controlled acceleration function. Training happened just before a car reset was required.
  Observations were taken over 4 frames, stacked, and input into the neural network.
  A 1D convolution and pooling were applied to this input, followed by a reduced complexity dense neural network.
- Findings: Reward shaping with 0.5 * speed at every time step for the first 1500 episodes incentivized speed initially.
  For episodes 1500-2000, the reward at every time step was set to 0 to shift the focus towards safety without encouraging constant acceleration.
  This phased approach helped the model balance speed and safety, leading to better performance in completing laps with fewer crashes.

B. Putting It All Together

While each modification addressed specific challenges in the autonomous vehicle speed control system, none provided a comprehensive solution to achieve consistently competitive lap times and safe driving behavior. Challenges such as turn anticipation, efficient braking, and robustness to simulator issues persisted across different modifications. Further research and experimentation are needed to develop a more robust and adaptive speed control system capable of meeting performance and safety requirements in dynamic environments.

V. Conclusion

In this project, we explored the application of Deep Q-Network (DQN) for optimizing the speed of an autonomous vehicle within a realistic, simulated environment. Our approach was driven by the need to ensure continuous operation, balance the competing demands of speed and safety, and maintain robust performance under varying conditions. The comprehensive architecture, which integrated a Unity-based simulator, the Robot Operating System (ROS), and a carefully designed reinforcement learning (RL) model, facilitated this endeavor. Our most notable modifications included the use of convolution neural networks (CNNs) to better process observation data and the implementation of curriculum training to sequentially teach the vehicle to prioritize speed and then safety.

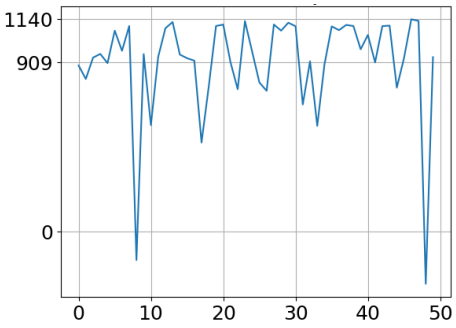

Fig. 4. A graph representing the cumulative rewards per episode on the training set

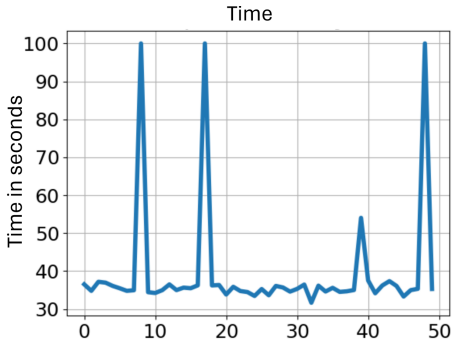

Fig. 5. Graph representing the test time taken per episode

In our evaluation, we found that the reinforcement learning model performed comparably to the manual PID controller. Specifically, while the manual PID controller took 32 seconds to complete 2 laps, our RL model took 36 seconds. This result indicates that our RL approach is nearly as effective as the manually optimized control system. Moreover, our model was able to adapt and improve its driving behavior through training, showcasing the potential for further enhancements.

The project faced several challenges, including handling the real-time requirements of training and execution, balancing the reward system, and preventing the model from learning sub-optimal behaviors. Despite these challenges, the overall performance of our RL model highlights the viability of using reinforcement learning for autonomous vehicle speed optimization.

Key Achievements:

Continuous Operation: We modified the standard DQN implementation to train the model only after completing an entire lap, thus preventing interruptions in real-time decision-making. This modification ensured that the vehicle received uninterrupted instructions, significantly reducing the risk of crashes due to delayed processing.

Reward Structuring and Curriculum Training: Our initial reward structure incentivized high speeds, which led to reckless driving. By implementing curriculum training, we first taught the car to race at high speeds and then focused on safety. This phased approach, coupled with early stopping, allowed us to develop a model that balanced speed with safety, achieving competitive lap times without compromising on crash avoidance.

Independence from Steering Control: By managing steering through a PID controller and excluding it from the neural network's considerations, we ensured that speed optimization decisions were unbiased by steering inputs. This separation allowed the neural network to concentrate on optimizing speed based solely on environmental inputs and previous actions.

Performance Evaluation: Extensive testing and iterative refinements across multiple cases highlighted the model's ability to learn and adapt. We saw significant improvements in lap times and driving safety as we progressed through various stages of model adjustments and training methodologies.

The project demonstrated the feasibility and effectiveness of using DQN for real-time autonomous vehicle speed optimization. The modifications we made to the traditional DQN approach were crucial in addressing the unique challenges posed by our simulation environment. The use of a realistic Unity simulator integrated with ROS provided a robust platform for testing and refining our model.

VI. Limitations and Future Work

Due to time constraints, we were only able to explore a limited scope within this project. The training of the model was time-intensive, taking approximately 23 hours.

To enhance the model's performance and efficiency, future work could explore:

- Incorporating more sophisticated neural network architectures to better handle the complexities of the driving environment.
- Enhancing the reward structure to dynamically balance speed and safety more effectively.
- Extending the simulation scenarios to include more diverse and challenging environments to ensure broader applicability of the model.
- Exploring advanced algorithms such as Actor-Critic (A2C) and Deep Deterministic Policy Gradient (DDPG) to potentially achieve better learning and performance outcomes.[11][12]

Acknowledgment

Elton and Yamini thanks Professor Changyou Chen for their invaluable guidance, support, and encouragement throughout the duration of this research project. Their expertise and mentorship have been instrumental in shaping our understanding of autonomous vehicle control systems and reinforcement learning techniques.

Furthermore, we would like to acknowledge the support and resources provided by IEEE, ResearchGate, which facilitated the execution of this research endeavor

References

[1] Keshav Bimbraw, "Autonomous Cars: Past, Present and Future A Review of the Developments in the Last Century, the Present Scenario and the Expected Future of Autonomous Vehicle Technology," Thapar University, P.O. Box 32, Patiala, Punjab, India, January 2015.

[2] Siegwart, R., Nourbakhsh, I. R. (2004), Introduction to Autonomous Mobile Robots, 2nd ed., MIT Press.

[3] Schwarting, W., Alonso-Mora, J., Rus, D. (2018), "Planning and Decision-Making for Autonomous Vehicles," Annual Review of Control, Robotics, and Autonomous Systems, 1(1), pp. 187-210.

[4] Pathfinding with Rapidly-Exploring Random Tree, https://rileyknox.github.io/rrt/.

[5] Mustafa Demir, Volkan Sezer. (2017) "Improved Follow the Gap Method for obstacle avoidance". 2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)

[6] Talwar, Palak. (2021). "AUTONOMOUS VEHICLE SAFETY OVERVIEW". University of Toronto's Self-Driving Car Team. Volume:03. pp. 2582-5208.

[7] B Ravi Kiran1. (2021) "Deep Reinforcement Learning for Autonomous Driving: A Survey". arxiv. Cornell University.

[8] R. S. Sutton and A. G. Barto. (2018). "Reinforcement Learning: An Introduction". (Second Edition). MIT Press. 1, 4, 5, 6

[9] Y. Quek, L. Koh, N. Koh, W. Tso, W. Woo. (2021). "Deep Q-network implementation for simulated autonomous vehicle control".

[10] Yesmina Jaafra, Jean Luc Laurent, Aline Deruyver, Mohamed S. Naceur. (2019). "Robust Reinforcement Learning For Autonomous Driving". ICLR 2019 Workshop on Deep RL Meets Structured Prediction.

[11] Noureldin Ragheb, Mervat M. A. Mahmoud. (2004). "Implementing Deep Reinforcement Learning in Autonomous Control Systems". The British University in Egypt. Journal of Advanced Research in Applied Sciences and Engineering Technology 41, Issue 1. pp. 168-178

[12] Lu Wen, Jingliang Duan, Shengbo Eben Li, Shaobing Xu, Huei Peng. (2020). "Safe Reinforcement Learning for Autonomous Vehicles through Parallel Constrained Policy Optimization". arxiv. Cornell University.