Course: CSE440
Section: 1
Group: 10

Israt Asha 1921758042
Abdullah Al Afique 2031117642
Tanha Ahmed Nijhum 2031523642

Self-Driving AI Car Racing Game

Abstract

This project presents a simulation-based approach to the development of a self-driving car system utilizing NEAT (NeuroEvolution of Augmenting Topologies) and Pygame. A virtual car learns to navigate complex racetrack maps using only five radar-based sensors and evolves through generations based on a fitness function. The solution demonstrates the application of evolutionary neural networks in autonomous driving, focusing on sensor input processing, map-based navigation, and reinforcement learning without supervised data.

Index Terms

NEAT, Self-driving car, Evolutionary learning, Python, Pygame, Reinforcement learning, Autonomous systems.

I. Introduction

Autonomous vehicles represent a rapidly growing field in artificial intelligence. Traditional methods often require large labeled datasets and high computational resources. However, NEAT offers a biologically inspired alternative: evolving neural networks via genetic algorithms. This project employs NEAT to train AI agents (cars) that learn to navigate through a virtual racetrack without any prior driving data, using fitness scores derived from performance metrics.

The goal is to explore the ability of neuroevolution to develop adaptive behavior in an environment where direct programming of optimal behavior is complex or impossible. With applications ranging from robotic motion control to real-time decision making, neuroevolution holds the promise of robust, general-purpose AI.

II. Related Work

Neuroevolution, especially through NEAT, has been widely used in control systems, video game AI, and robotics. The 2002 paper by Stanley and Miikkulainen introduced the NEAT algorithm,

emphasizing its potential to evolve both topology and weights of neural networks simultaneously. This approach contrasts traditional gradient-based learning methods that require fixed architectures.

III. System Design

A. Tools and Libraries

- **Python 3**: Main programming language
- **Pygame**: For rendering the graphical simulation
- **NEAT-Python**: Implementation of NEAT for network evolution
- **NumPy and Math Libraries**: For calculations and sensor modeling

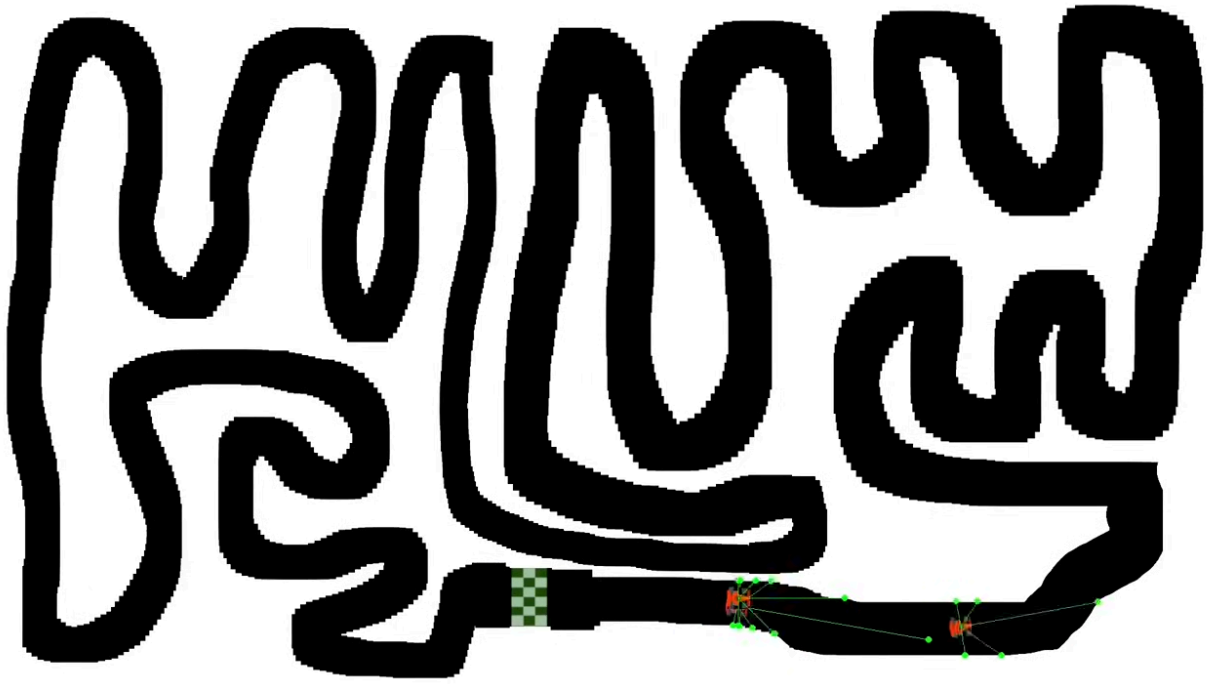
B. Components

1. **Car Class**:
    - Handles movement, collision, radar sensors, and rendering.
    - Sensors emit rays at angles to detect borders.
    - Collision occurs if any sensor detects the white border color.
2. **Simulation Loop**:
    - Initialized with multiple genomes from the population.
    - Each genome is used to create a neural network and corresponding car.
    - Each car receives sensor data and takes actions (steering, acceleration) based on network outputs.
3. **NEAT Configuration**:
    - Configured via `config.txt` file.
    - Key parameters: population size, mutation rate, elitism, input/output nodes.

4. **Fitness Reporting**:
    - NEAT provides tracking of average, max, and stagnated genomes.
    - Simulation outputs are logged and visualized per generation.

C. Car Mechanics

Each car is a sprite with basic physics applied. It responds to speed, angle, and radar input. Rotations, distance traveled, and alive status are continuously evaluated. Collisions are checked against a color-coded map (white = walls).



IV. Methodology

A. Sensor Data as Input

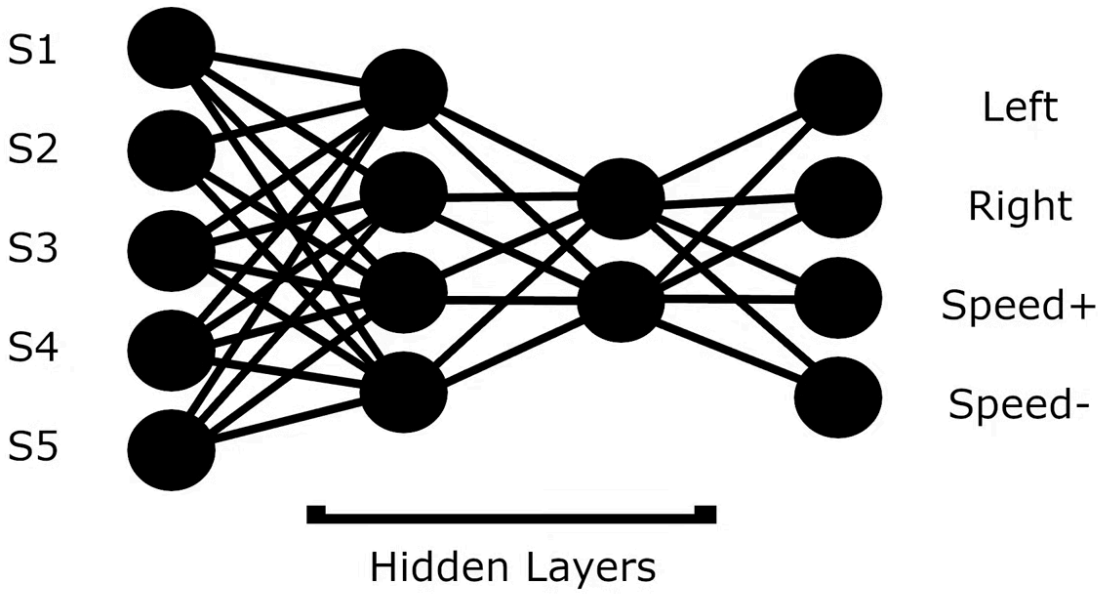
Each car has 5 radars, returning distances to the track boundary. These distances form a 5-dimensional input vector. The sensors are spaced with angle steps (e.g., -90°, -45°, 0°, 45°, 90°), simulating a vision cone.

B. Action Selection via Network Output

Neural network outputs:

- Index 0: Steer Left
- Index 1: Steer Right
- Index 2: Decrease Speed
- Index 3: Increase Speed

Only one action is chosen per tick using the max-activation index.



C. Fitness Function

Cars are rewarded based on the distance traveled without crashing:

Fitness=DistanceCAR_SIZE_X/2\text{Fitness} = \frac{\text{Distance}}{\text{CAR_SIZE_X} / 2}

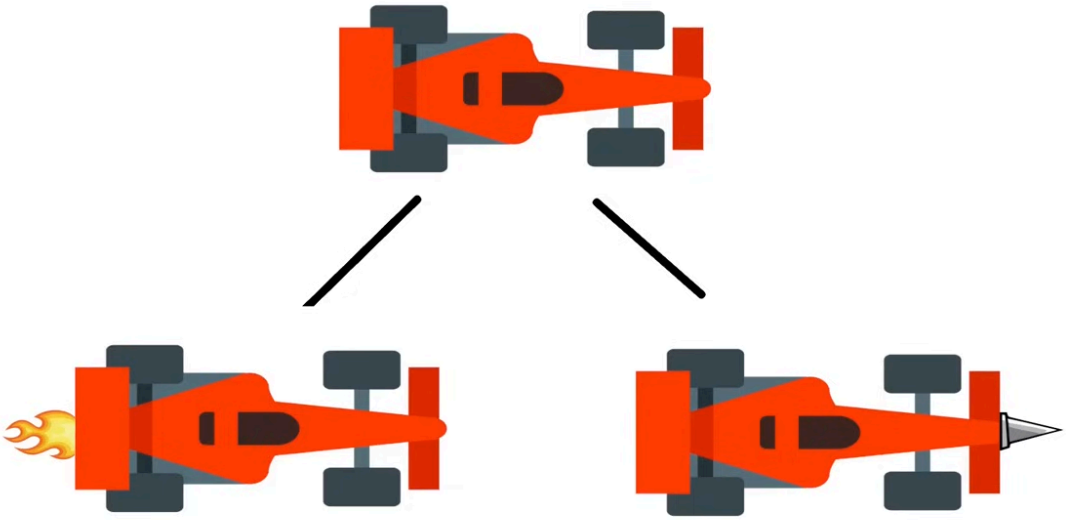
This incentivizes continuous forward movement while penalizing erratic or risky turns that result in collisions.

D. Evolution Strategy

After each generation:

- Top-performing cars reproduce.
- New generations are created via crossover and mutation.
- Poor performers are discarded.

Species are tracked to preserve innovation and avoid premature convergence.



V. Experimental Setup

A. Environment

- **Map Dimensions**: 1920x1080 pixels
- **Car Size**: 60x60 pixels
- **Sensors**: 5 angles from -90 to 90 degrees
- **Max Simulation Time**: ~20 seconds per generation (adjustable)
- **Generations**: Up to 1000

- **Initial Speed**: 20 (adjusted via output neurons)

B. Parameters

- **Population Size**: 30
- **Max Fitness Threshold**: Uncapped
- **Mutation Probabilities**: Tuned via `config.txt`
- **Activation Function**: Default sigmoid

C. Testing Maps

1. Simple Ring Track
2. Curved Racetrack
3. Advanced Winding Maze
4. Chaotic Stroke-Like Maze (hardest)

Each map was designed with increasing complexity and varying path widths.

VI. Results

A. Basic Circular Map

- Within the first few generations, several cars complete laps.
- Simple steering behavior (left-right decisions) evolved rapidly.
- Reproduced behavior persisted through generations.

B. Intermediate Map with Curves

- Took ~7 generations for two cars to complete the track.

- Variety emerged: slow but steady vs. fast and risky.
- NEAT maintained diverse species to allow experimentation.

C. Complex Map

- Required manual tuning: hidden layers removed, fixed speed enforced.
- Focused evolution on steering decisions only.
- Eventually, successful completion of a highly complex track.

D. Performance Trends

- Early generations show random movement.
- Later generations show adaptation and learning.
- Species mechanism of NEAT preserves innovation.

VII. Visualizations and Debugging Tools

- **Radar Lines**: Visualize sensor detection and boundaries
- **Alive Count & Generation Display**: Runtime metrics
- **Fitness Graphs**: Optional for tracking evolution
- **Collision Zones**: Mapped with color-coded feedback

Visual feedback played a major role in debugging and model verification.

VIII. Discussion

A. Design Trade-offs

- Using fixed speed simplified the learning process.
- Reducing output complexity improved performance on hard tracks.
- Fitness shaping (reward tuning) played a major role.

B. Limitations

- Simulation only operates in 2D.
- No dynamic obstacles or real-time sensors.
- Training is CPU-intensive for large generations.

C. Future Improvements

- Add traffic or dynamic obstacles.
- Extend to 3D environments with more realistic physics.
- Integrate real-world driving datasets for hybrid learning.
- Incorporate attention-based models for input filtering.

IX. Conclusion

This project successfully demonstrates the application of NEAT for autonomous driving simulations. By evolving neural networks based on radar sensor input and reward feedback, the system shows strong adaptability across varying track complexities. It highlights how evolutionary learning can be used for behavior emergence without relying on supervised learning or deep models.

NEAT's flexible topology and species-based reproduction strategy proves to be an effective approach to evolve robust control policies in real-time simulations. With proper tuning, even complex mazes can be solved through repeated cycles of mutation and selection.

X. Acknowledgements

This project builds upon the contributions of the AI open-source community. Special thanks to:

- **Cheesy AI** for the original concept
- **NeuralNine (Florian Dedov)** for optimizations and documentation
- **NEAT-Python team** for maintaining the core library
- **YouTube and GitHub communities** for inspiration and peer feedback

References

[1] S. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99-127, 2002.

[2] Florian Dedov (NeuralNine), "Self-Driving AI Car Simulation in Python," YouTube, https://www.youtube.com/watch?v=Cy155O5R1Oo.

[3] Cheesy AI, "AI Driving Simulation in Python," GitHub Repository (Base Version).

[4] NEAT-Python Library, https://github.com/CodeReclaimers/neat-python

[5] Pygame Documentation, https://www.pygame.org/docs/

[6] J. Clune, B. E. Beckmann, C. Ofria, and R. T. Pennock, "Evolving Coordinated Quadruped Gaits with the HyperNEAT Generative Encoding," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, 2008.
